




KeepCore
Conception & développement de logiciels
et applications mobiles
Montpellier, France

&

Université de Montpellier
Institut Universitaire de Technologie
Département Informatique
Montpellier, France

# Rapport de stage

## HandiCarParking




*Auteur :*
Gaël Foppolo

*Tuteurs :*
Jérémy Reynaud
Victor Poupet

Année universitaire 2014 - 2015

# Remerciements

Avant toute chose, je souhaite remercier l'équipe de KeepCore dans son ensemble pour la sympathie et l'accueil qui m'ont été accordés. Toutes ces personnes ont contribué au bon déroulement de ce stage.

Je voudrais particulièrement remercier Anne-Sophie et Patrick, pour l'aide et les remarques qu'ils m'ont apportées pendant l'écriture du rapport.

Enfin, je tiens à remercier Jérémy pour le temps qu'il m'a consacré tout au long du stage. Très ouvert d'esprit, sa patience, sa disponibilité et ses conseils ont été d'une aide précieuse et vraiment apprécié tout au long de ce stage.

# Table des matières

# Table des figures

# Acronymes

**GIG-GIC** grand invalide de guerre - grand invalide civil. 1

# Glossaire

**App Store** la plateforme officielle de téléchargement d'applications en ligne pour les appareils iOS. 1

**gesture** mouvement d'un ou de plusieurs doigts qui se produit sur une zone spécifique de l'écran et qui permet d'interagir avec l'application. 13, 15, 27, 43, 45

**internationalisation (i18n)** processus visant à modifier l'application, lui permettant ainsi le support de plusieurs langues. 36

**liaison optionnelle** on utilise la liaison optionnelle ou optional binding pour savoir si une optionnelle contient une valeur et si c'est le cas, faire en sorte que sa valeur soit disponible de manière temporaire. 28

**localisation (l10n)** processus visant à ajouter les ressources nécessaires à l'application, afin qu'une langue soit prise en charge, comme l'Espagnol. 36

**open data** données d'origines publiques ou privées, diffusées de manière structurée et sous licence libre, garantissant son libre accès et sa réutilisation sans aucune restriction. 1, 53

**optionnelle** une variable peut-être optionnelle c'est-à-dire contenir une valeur ou ne pas en contenir (nil). 28

**refactoring** retravailler le code source d'un programme informatique, pour le rendre plus générique ou améliorer la lisibilité, sans toutefois y ajouter des fonctionnalités ou corriger des bugs. I

**swipe** mouvement de gauche à droite sur l'écran avec un seul doigt, permettant de retourner à l'écran précédent. 45

CHAPITRE 1

# Introduction

On estime que près de 15% de la population mondiale serait aujourd'hui touchée par un handicap sous une forme ou une autre, soit près d'un milliard de personnes [1]. Et ce chiffre est en hausse constante, en raison du vieillissement grandissant de la population et de la propagation de plus en plus rapide des maladies chroniques.

Dans le monde entier, les personnes handicapées ont des difficultés pour accéder à des services que beaucoup d'entre nous considèrent depuis longtemps comme des acquis, dans les domaines de la santé, de l'éducation, de l'emploi ou des transports. Le manque d'accès au transport est, pour une personne handicapée, un motif fréquent la dissuadant de chercher du travail ou l'empêchant d'accéder aux soins de santé. Il existe de nombreuses places de parking disponibles pour les personnes handicapées, que ce soit en ville ou dans de plus petits villages. En Europe, les places pour personne handicapée, matérialisées par un fauteuil roulant, sont réservées aux seuls titulaires de la carte européenne de stationnement [2], qui remplace maintenant les macarons GIG-GIC.

Mais dans de nombreux cas, les systèmes de transport et l'information ne sont pas accessibles à tous. Il y a peu d'informations disponibles sous une forme facile d'accès et de nombreux besoins ne sont pas couverts pour les personnes handicapées.

L'application *HandiCarParking* a pour objectif de faciliter l'accès aux places de parkings réservées aux personnes en situation de handicap dans le monde entier. Elle permet de visualiser les emplacements de places sur une carte et obtenir des informations complémentaires y étant associées. Elle permet également d'obtenir la prise en charge de l'itinéraire via des applications tierces et la visualisation du panorama de l'emplacement.

Dans le cadre de ce projet, la plateforme ciblée est uniquement iOS. L'application se propose d'utiliser le projet open data, *OpenStreetMap* comme source de données et *Google Maps* pour le rendu de ses données. *OpenStreetMap* est un projet ayant pour but de créer une base de données géographique libre et mondiale et peut se comparer à un « Wikipedia pour cartes ». Les données collectées sont très variées, de l'emplacement des pylônes électriques à la superficie des forêts en passant par la largeur des trottoirs.

Le but du stage est de réaliser une étude préliminaire des besoins afin de dégager les grandes lignes en tenant compte des contraintes. Il s'agit ensuite de formaliser les résultats de cette étude sous la forme de spécifications puis de développer l'application en elle-même. L'objectif final est de publier l'application sur l'App Store.

Chapitre 2

# Présentation de l'entreprise

## 2.1 KeepCore




Figure 2.1 – Logo de KeepCore

Située au cœur de Montpellier, KeepCore est une SSII créée en 2004 par Jérémy Reynaud et Patrick Carias, encore actuels dirigeants. Pour l'exercice 2013-2014, son chiffre d'affaires est d'environ 750 000€ pour un résultat avoisinant les 100 000€. La principale activité de KeepCore est la conception et le développement de logiciels et applications mobiles. L'entreprise est composée d'une petite équipe de 8 personnes, dont 6 ingénieurs (voir figure 2.2).

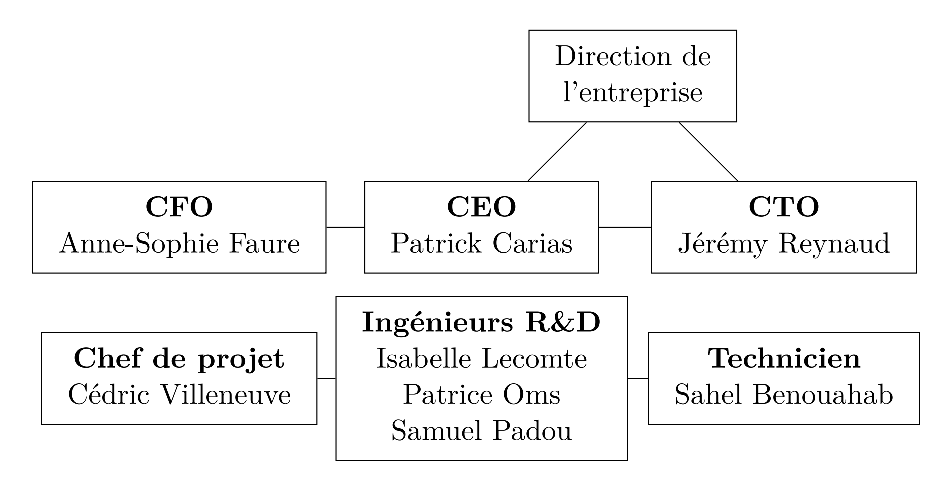



Figure 2.2 – Organigramme de KeepCore

## 2.2 Activité

KeepCore développe tous types d'applications que ce soit web, bureau, mobile ou tablette, mais d'un genre particulier : elle vise le développement d'applications répondant à des besoins spécifiques, généralement peu ou mal adressés par les solutions du marché. De ce que j'ai pu voir pendant mon stage, les projets touchent des domaines très variés : appels d'offres, système de paiements, solutions médicales, etc. Ce large éventail de projets nécessite une grande pluralité de compétences, autant dans les aspects métier (la compréhension des besoins du client) que dans les technologies utilisées : GWT, Spring, AngularJS, J2EE, etc. C'est cette facette qui m'a le plus marqué durant le stage, voir comment chacun s'adapte à la situation et apprend chaque jour à développer une nouvelle compétence pour répondre aux besoins.

#### 2.2.0.1 Exemple de projet

**Espace Rendez Vous** est une solution de prise de rendez-vous en ligne et de gestion de ressources ultra flexible, qui équipe aujourd'hui plusieurs grandes entreprises. Les principales fonctionnalités couvertes par la solution sont :

- gestion des ressources matérielles : disponibilité, rotation, réservation, etc.
- statistiques d'utilisation des ressources humaines et matérielles
- notifications des clients par SMS
- gestion du planning de l'équipe
- etc.

## 2.3 Clients

KeepCore met ses services à disposition de clients de toutes tailles : startups, éditeurs, grands comptes, PME, collectivités, etc. (voir figure 2.3)
Elle opère avec des clients d'horizons différents et s'adapte en fonction des besoins, des processus métiers ou des contraintes.

## 2.4 Équipe

L'équipe de KeepCore est dirigée par Jérémy Reynaud et Patrick Carias. Jérémy en tant que directeur technique, s'occupe des questions techniques étant lui-même ingénieur. Patrick s'occupe de la relation client, du marketing ainsi que d'une partie de l'administratif. Anne-Sophie quand à elle, s'occupe de la partie administrative et financière de l'entreprise. Sahel s'occupe d'effectuer les tests sur les produits développés par KeepCore. Isabelle, Cédric, Patrice et Samuel s'occupent de l'analyse, la recherche, la conception, le développement et le déploiement des solutions.
Isabelle, Sahel, Cédric, Patrice et Samuel sont actuellement en mission chez des clients (voir section 2.6), certains à plein temps d'autre seulement à mi-temps.

## 2.5 Prise de décision

Chaque lundi matin, les trois directeurs de KeepCore, Patrick, Jérémy et Anne-Sophie, se réunissent lors du comité de direction hebdomadaire. Pour gérer les sujets à traiter et garder une trace écrite, un historique de ces comités de direction, ils utilisent un outil interne, développé

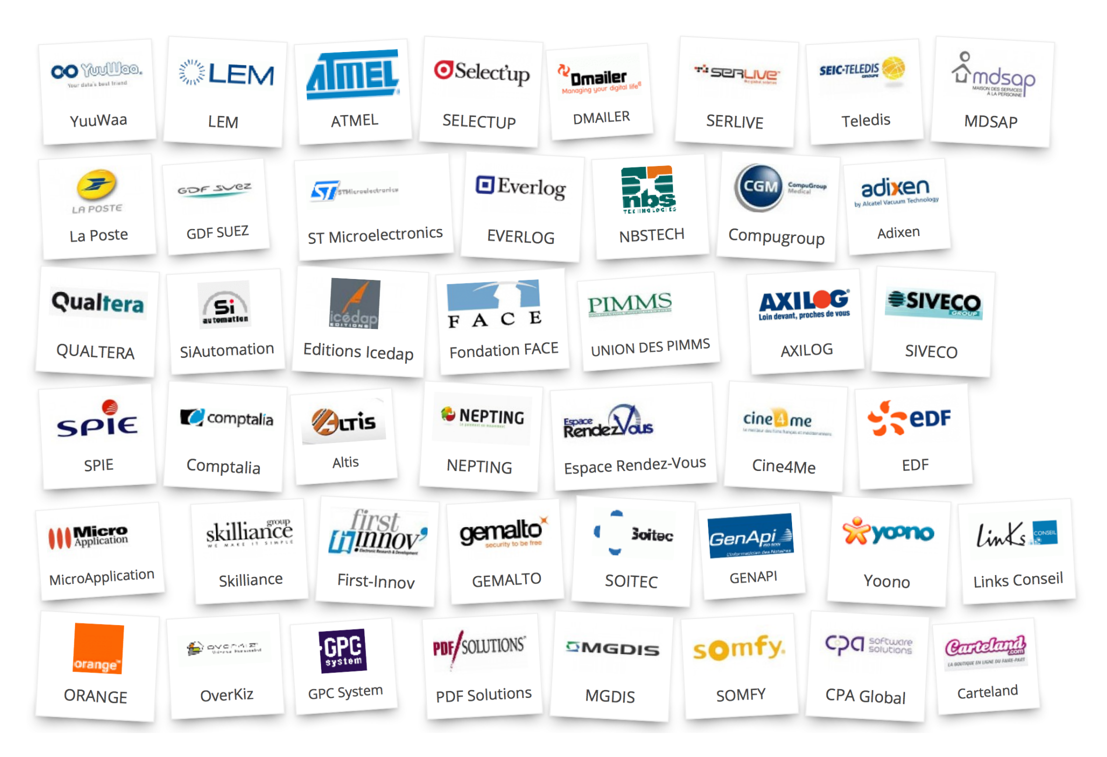
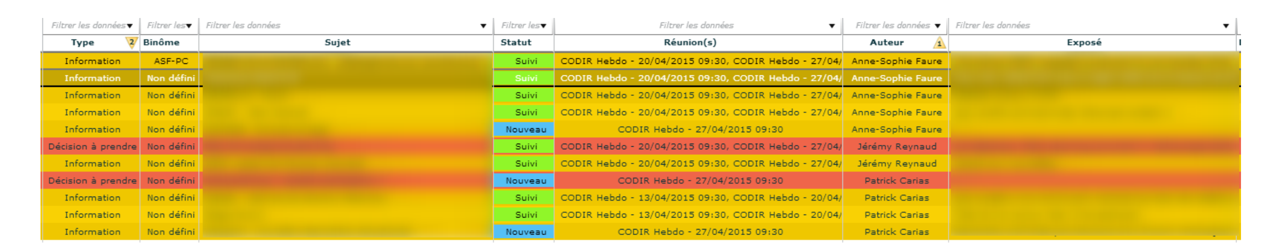



Figure 2.3 – Clients de KeepCore

par KeepCore : KeePlace (voir figure 2.4).



| Type | Binôme | Sujet | Statut | Réunion(s) | Auteur | Exposé |
|---|---|---|---|---|---|---|
| Information | ASF-PC | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Anne-Sophie Faure | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Anne-Sophie Faure | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Anne-Sophie Faure | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Anne-Sophie Faure | [illegible] |
| Information | Non défini | [illegible] | Nouveau | CODIR Hebdo - 27/04/2015 09:30 | Anne-Sophie Faure | [illegible] |
| Décision à prendre | Non défini | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Jérémy Reynaud | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 20/04/2015 09:30, CODIR Hebdo - 27/04/ | Jérémy Reynaud | [illegible] |
| Décision à prendre | Non défini | [illegible] | Nouveau | CODIR Hebdo - 27/04/2015 09:30 | Patrick Carias | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 13/04/2015 09:30, CODIR Hebdo - 20/04/ | Patrick Carias | [illegible] |
| Information | Non défini | [illegible] | Suivi | CODIR Hebdo - 13/04/2015 09:30, CODIR Hebdo - 20/04/ | Patrick Carias | [illegible] |
| Information | Non défini | [illegible] | Nouveau | CODIR Hebdo - 27/04/2015 09:30 | Patrick Carias | [illegible] |

Figure 2.4 – KeePlace - Comité de direction

Chacun, pendant 20 min, prend la parole et expose ses sujets, afin de pouvoir en débattre avec les autres. Ses sujets se classent en deux catégories : **information** et **décision à prendre**.

Au début de chaque exercice (en octobre pour KeepCore), un chiffre d'affaires souhaité est déterminé, c'est l'objectif à atteindre. Lors de ce comité, ils effectuent un « bilan » de la semaine, extrapolent le chiffre d'affaires du mois en fonction du chiffre d'affaires actuel (jour du comité) et vérifient si cela correspond bien au chiffre d'affaires souhaité du mois. Des décisions sont prises en fonction de ce résultat.

D'autres rares réunions sont effectuées selon les besoins, mais les décisions principales sont prises lors de ce comité.

## 2.6 Gestion des projets

Le contrat qui formalise la collaboration entre une SSII et une entreprise peut être de 2 types : **l'engagement de résultat** ou **l'engagement de moyens**.

L'**engagement de résultat** est un engagement contractuel d'une SSII sur un résultat attendu. Ainsi, lors de la signature d'un accord entre client et SSII, la SSII s'engage à fournir une prestation et des livrables. Ce type d'engagement ne laisse pas la place aux imprévus.

**L'engagement de moyens** consiste à faire payer le temps réellement passé pour le service, sans obligation de résultats.

KeepCore utilise la méthode Scrum pour gérer ses projets et par conséquent n'effectue presque plus d'engagement de résultat, trop rigide. Pour gérer les projets, plusieurs outils sont utilisés en plus des réunions journalières et hebdomadaires entre développeurs et avec le client :

- **JIRA** : système de suivi de bugs, gestion des incidents et gestion de projets
- **Gantt Project** : planification d'un projet à travers la réalisation de diagrammes de Gantt (voir figure 2.5)
- **KeePlace** : outil interne pour le suivi du temps de travail par projet

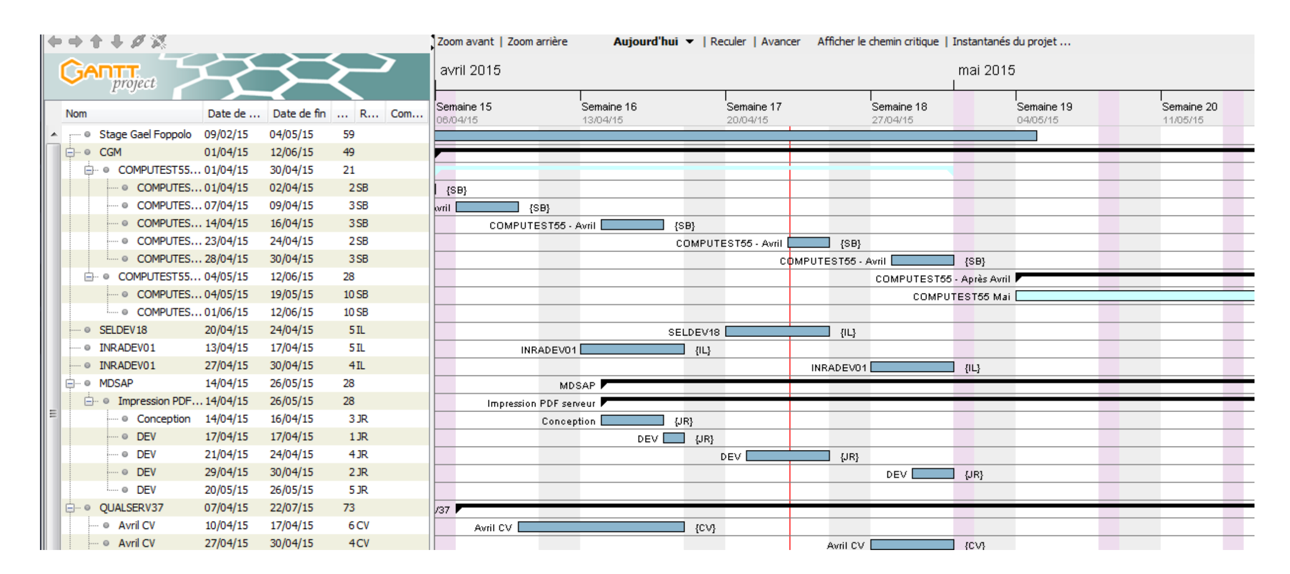



Figure 2.5 – Gantt Project

Chapitre 3

# Cahier des charges

## 3.1 Introduction

### 3.1.1 Objectifs

Cette partie présente les spécifications fonctionnelles et non fonctionnelles du projet *HandiCarParking*. Il peut être utilisé comme base pour une possible extension du projet actuel. Ce cahier des charges est rédigé sous le standard *IEEE 830* [3].

### 3.1.2 Public visé

Cette partie ainsi que la suivante sont destinées :

- aux développeurs qui peuvent examiner les capacités du projet et plus facilement comprendre où leurs efforts devraient être ciblés pour améliorer ou ajouter des fonctionnalités au projet (il établit les lignes directrices pour un développement futur)
- aux testeurs qui peuvent utiliser cette partie comme base pour leur stratégie de tests, certains bugs peuvent être plus faciles à déceler à l'aide de ces documents
- aux utilisateurs finaux de l'application qui souhaitent en savoir plus sur ce que ce projet peut faire

## 3.2 Description générale

Cette section donnera un aperçu de l'ensemble du système. Le système sera expliqué dans son contexte pour montrer comment celui-ci interagit avec les utilisateurs et les fonctionnalités principales seront introduites. Enfin, les contraintes et les hypothèses pour le système seront présentées.

### 3.2.1 Perspective du produit

Le système va devoir communiquer avec l'utilisateur afin de récupérer les données nécessaires à son fonctionnement, les traiter et enfin afficher les résultats. Il devra être capable de récupérer sa position actuelle dans le monde ainsi que de récupérer un nom de localité. L'application *HandiCarParking* ne fonctionne pas de façon indépendante. Comme c'est une application centrée sur les données, elle devra communiquer avec différents services externes tels qu'*OpenStreetMap*. Aucune donnée ne sera stockée de façon persistante. Toutes les communications se feront par **Internet**.

Le système de l'application est composé d'un seul acteur et trois systèmes qui coopèrent. L'utilisateur de l'application est notre seul acteur. Le système principal (gestion de la géolocalisation, traitement des données, gestion de l'interface, etc.), *OpenStreetMap* et *Google Maps* sont les 3 systèmes qui coopèrent.

Chaque acteur accède au système principal et communique avec les deux autres systèmes via l'interface utilisateur que lui fourni celui-ci (voir figure 3.1). Dans la suite du cahier des charges, les acteurs seront désignés par le terme **utilisateurs** et le système de l'application de notre projet sera désigné par le terme **application**.

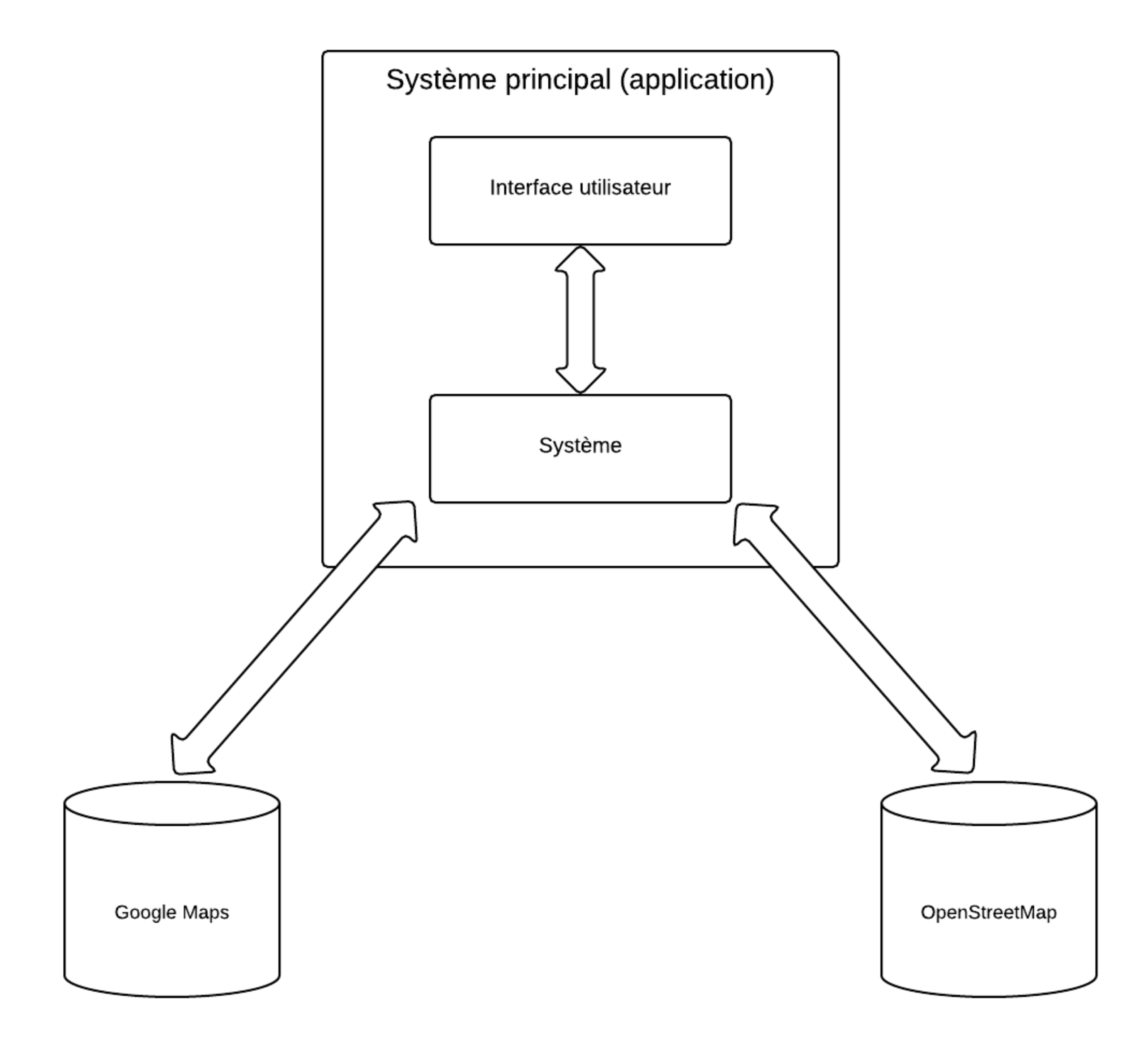



FIGURE 3.1 – Schéma fonctionnel (block diagram)

### 3.2.2 Caractéristiques des utilisateurs

Ce projet ne se réfère qu'à un seul type d'utilisateur dans sa première version : la **personne en situation de handicap**.
Dans les versions ultérieures, de nouveaux types d'utilisateurs pourront être ajoutés : administrateurs, modérateurs, contributeurs. etc. L'utilisateur interagit avec l'application via l'interface de celle-ci, plus communément appelé **vue** ou **fenêtre**.
Cet utilisateur attend de l'application qu'elle soit rapide, intuitive et accessible. L'utilisateur recherche avant tout la simplicité lors de son expérience, l'utilisation d'icônes accompagnées de petits labels sont à préférer pour offrir une meilleure clarté de l'information, de même que l'absence d'options superflues, obligeant l'utilisateur à réfléchir et donc contre intuitif.

### 3.2.3 Environnement de fonctionnement

*HandiCarParking* est une application conçue pour le système mobile d'*Apple*, *iOS*. Seule la version du système pour *iPhone* est supportée. La version minimale de support est **iOS 8.0**, ce qui actuellement ne convient qu'aux **iPhone 4S et iPhone ultérieurs**.
L'application requiert un accès à Internet, l'activation du service de géolocalisation de l'appareil et l'autorisation pour l'utilisation de ce service par l'application.

L'application doit être compatible avec tous les *iPhone* supportant iOS 8.X, c'est-à-dire :

- iPhone 4S
- iPhone 5
- iPhone 5C
- iPhone 5S
- iPhone 6
- iPhone 6 Plus

### 3.2.4 Fonction du produit

L'application doit offrir **trois fonctionnalités principales** à un utilisateur :

- afficher les places de parkings réservées aux personnes handicapées autour de sa **position actuelle**
- afficher les places de parkings réservées aux personnes handicapées autour d'un **lieu de son choix**
- offrir la possibilité d'une prise en charge d'un **itinéraire** par une application tierce

Le système principal doit interagir avec l'utilisateur pour récupérer sa position ou le lieu qu'il a choisi, effectuer les actions nécessaires auprès des services externes afin de récupérer les emplacements des places de parkings et les afficher sur l'interface utilisateur. Il doit aussi gérer les cas d'erreurs lors d'une tentative d'accès à une partie de l'application si tous les prérequis ne sont pas satisfaits, afin de toujours rester dans un état cohérent. La figure 3.2 détaille les fonctionnalités auxquelles l'utilisateur doit avoir accès.

#### 3.2.4.1 Se géolocaliser

L'utilisateur doit avoir la possibilité de se géolocaliser et de connaître sa position, sous la forme d'un marqueur sur la carte.

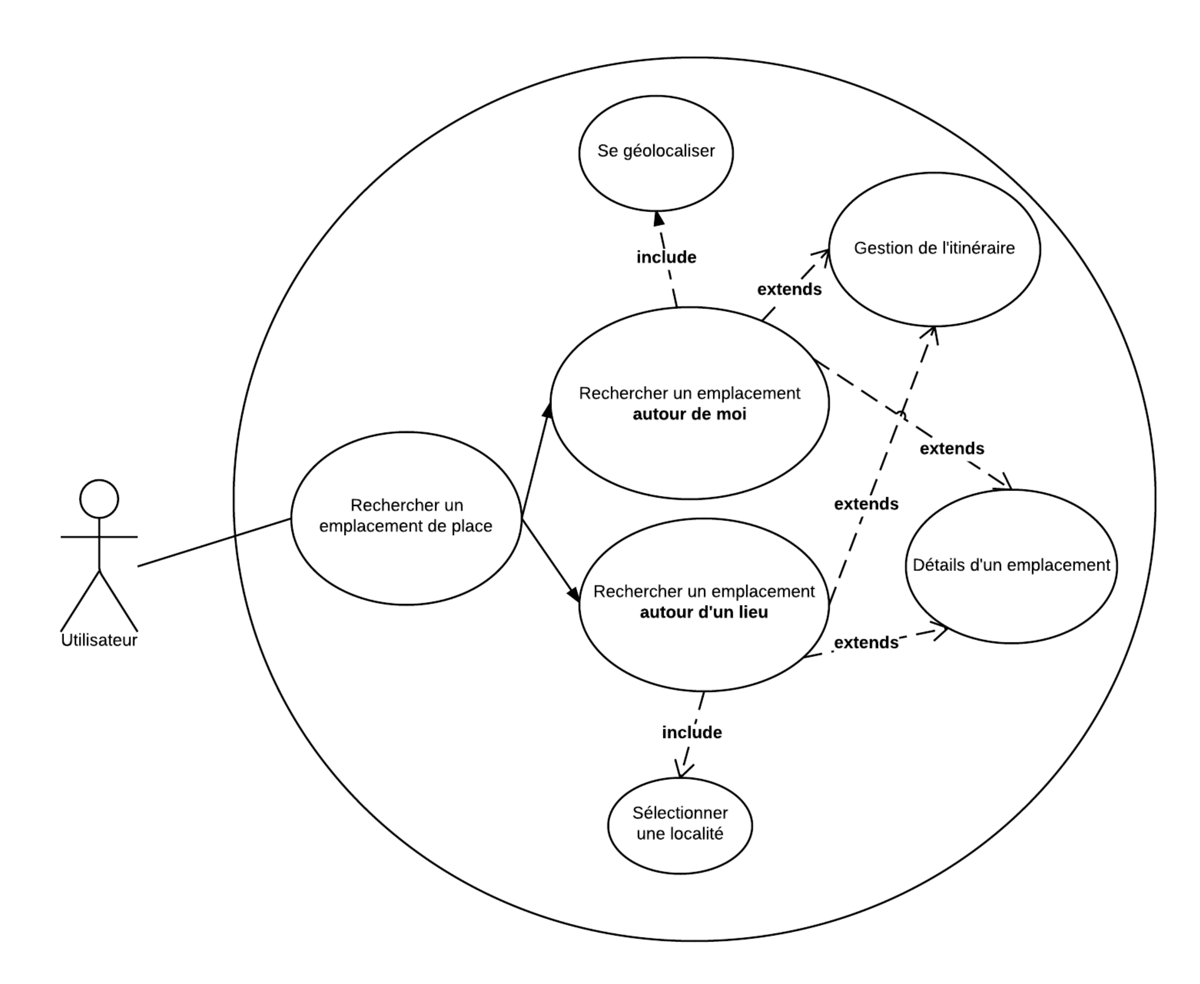



FIGURE 3.2 – Diagramme des cas d'utilisations

#### 3.2.4.2 Sélectionner une localité

L'utilisateur doit avoir la possibilité de rechercher une localité que ça soit par son nom, son code postal ou son adresse. Il doit avoir une liste du ou des résultats obtenus ou une liste vide.

#### 3.2.4.3 Rechercher les places autour de moi

L'utilisateur doit avoir la possibilité de rechercher les emplacements des places de parkings réservés aux personnes handicapées proche de sa position. Les résultats de cette recherche doivent être affichés sous forme de marqueurs sur la carte.

#### 3.2.4.4 Rechercher les places autour d'une localité

L'utilisateur doit avoir la possibilité de rechercher les emplacements des places de parkings réservés aux personnes handicapées proche d'une localité qu'il aura préalablement choisie. Les résultats de cette recherche doivent être affichés sous forme de marqueurs sur la carte.

#### 3.2.4.5 Afficher les détails d'un emplacement

L'utilisateur doit avoir la possibilité d'afficher les détails d'un emplacement de place de parking, par exemple si la place est payante ou non.

### 3.2.5 Contraintes

L'une des contraintes imposées lors du choix du projet est l'utilisation d'*open data* et notamment du projet *OpenStreetMap* (OSM). Les données extraites d'OSM sont disponibles à l'utilisation sous licence *ODbL* [4].

*HandiCarParking* est une application développée pour **iOS 8**, ce qui impose l'utilisation d'un langage, l'*Objective-C* ou le *Swift* ou une combinaison des deux. *KeepCore* a fait le choix de la nouveauté en choisissant le **Swift**. Le suivi de certaines règles de design édictées par *Apple* (voir annexe B) est une volonté de *KeepCore*.

L'application doit être visuellement identique, quel que soit le modèle d'*iPhone* utilisé, chaque *iPhone* n'ayant pas la même résolution. Le but est d'arriver à uniformiser au maximum le design de l'application pour qu'il y ait le moins de différences possible lors de l'affichage sur différents modèles.

La connexion **Internet** est aussi une exigence pour l'application. Celle-ci récupère des données en entrée, les traite à l'aide de services externes et génère les résultats. La géolocalisation est aussi une exigence, sans son activation, une grande partie des fonctionnalités n'est pas disponible.

### 3.2.6 Critères de succès

#### 3.2.6.1 Compatibilité

Une application *iOS* est conçue de façon à être complètement compatible et opérationnelle avec les futures versions d'*iOS*. Il n'y a pas d'opération supplémentaire à effectuer pour assurer cette post-compatibilité. En revanche, l'application doit supporter une version *iOS* minimale. Cette version minimale est assujettie aux fonctionnalités utilisées dans l'application et aux règles de compatibilité en vigueur lors de son développement. Les versions antérieures ne supporteront pas l'application.

#### 3.2.6.2 Disponibilité

*HandiCarParking* étant dépendant de services externes publics, sa disponibilité est tributaire de la disponibilité de ses services. Les services de *Google* étant populaires et très largement utilisés, il est très peu probable que ses services soient indisponibles. En revanche, les serveurs d'*OpenStreetMap* étant hébergés par des associations ou des particuliers, il est possible que ceux-ci soient indisponibles, lors d'une forte charge, par exemple.

#### 3.2.6.3 Sécurité

Aucune donnée sensible n'est manipulée mise à part la localisation d'utilisateur, aucune sécurité autre que celle intégrée par défaut au système n'est donc utile à intégrer.

Chapitre 4

# Spécifications fonctionnelles détaillées

## 4.1 Introduction

### 4.1.1 Objet du document

L'objet de ce document est de définir les spécifications fonctionnelles et techniques détaillées de l'application *HandiCarParking*. Les spécifications ont pour but de décrire précisément :

- l'ensemble des fonctionnalités de l'application
- les écrans utilisateurs mettant en œuvre les fonctionnalités de l'application
- le but, le type et le caractère obligatoire de chacun des champs présents sur les écrans de saisie, ainsi que les actions possibles à partir des écrans
- toutes les fonctionnalités prévues lors de la phase de conception sont précisées dans ce document en indiquant l'implémentation de ces fonctionnalités dans l'application
- des maquette d'écrans illustreront ce document

### 4.1.2 Fonctionnalités principales

#### 4.1.2.1 Emplacement(s) proche(s) de ma position

L'application permet à l'utilisateur de rechercher les emplacements de places de parkings réservés aux personnes handicapées autour de sa **position actuelle**. Pour cela, sa position est préalablement récupérée, avant de faire le traitement nécessaire pour récupérer les places. Lorsque les emplacements sont récupérés, ils sont affichés sur une carte sous la forme de marqueurs.

#### 4.1.2.2 Emplacement(s) proche(s) d'une localité

L'application permet à l'utilisateur de rechercher les emplacements de places de parkings réservés aux personnes handicapées autour d'**un lieu qu'il aura choisi**. Pour cela, il doit préalablement choisir le lieu, via une recherche intégrée. Lorsque son choix est fait, le traitement pour la récupération des places est effectué et les résultats sont affichés sur une carte sous la forme de marqueurs.

#### 4.1.2.3 Prise en charge de l'itinéraire

L'application permet à l'utilisateur d'être redirigé vers une application tierce, qui va gérer l'**itinéraire** entre sa position actuelle et la position de l'emplacement qu'il a choisi.

## 4.2 Analyse de l'existant

Il n'existe que très peu d'applications présentant des fonctionnalités similaires à *HandiCarParking*, néanmoins deux ont été choisies et analysées.

| Nom | Version | Publication | iOS mini | Compatibilité | Langue |
|---|---|---|---|---|---|
| `Handicap.fr` | 1.4 | avril 2012 | 4.0 | iPhone | Français |
| `Handi Parking` | 2.0 | janvier 2015 | 7.0 | iPhone, iPad | Français, Anglais |

### 4.2.1 Handicap.fr

| Nom | Version | Publication | iOS mini | Compatibilité | Langue |
|---|---|---|---|---|---|
| `Handicap.fr` | 1.4 | avril 2012 | 4.0 | iPhone | Français |

Propose l'indexation des places de parking réservées aux personnes handicapées dans toutes les localités françaises.

L'application a été testée sur **iOS 8.1.2** avec un **iPhone 5S** (6.2).

L'application possède quatre fonctionnalités principales qui sont disponibles depuis l'écran d'accueil de l'application :

- affichage des places proches de **ma position actuelle** grâce à la géolocalisation du téléphone (*Wi-Fi* ou *GPS*)
  - sous forme d'une carte avec possibilité d'obtenir l'adresse en touchant un marqueur
  - sous forme d'une liste classée selon la distance croissante
- recherche des places **autour d'un lieu** (ville, code postal ou adresse)
  - sous forme d'une carte
  - sous forme d'une liste
- liste des emplacements mémorisés (ou **favoris**)
- soumettre un **nouvel emplacement** non répertorié




FIGURE 4.1 – Handicap.fr - Place(s) proche(s) de moi

#### 4.2.1.1 Emplacement(s) proche(s) de ma position

Après avoir tapé sur le bouton, une nouvelle vue apparaît (voir figure 4.1), affichant une carte et intégrant les éléments suivants :

- notre position (un point bleu)
- des marqueurs numérotés (les emplacements de places, maximum 20)

- les *gestures* habituels de zoom, dézoome, rotation, etc.
- un bouton qui permet d'afficher les emplacements sous forme de liste

En effectuant un tap sur un marqueur, des informations complémentaires s'affichent :

- adresse, distance
- un bouton pour la prise en charge de l'itinéraire (seul *Plans* est disponible)
- un bouton pour le partage par e-mail
- un bouton pour la mémorisation de la place dans les favoris

#### 4.2.1.2 Emplacement(s) proche(s) d'une adresse




FIGURE 4.2 – Handicap.fr - Recherche

Après avoir tapé sur le bouton, une nouvelle vue apparaît (voir figure 4.2), demandant de renseigner les éléments suivants :

- ville
- code postal
- adresse

Si l'association n'est pas correcte (ville/code postal par exemple) on obtient une erreur, de même si la ville n'est pas dans la base de données de l'application.[1]

Après avoir renseigné la ville et/ou le code postal et/ou l'adresse, une nouvelle vue apparaît et affiche de la même façon que dans la fonctionnalité précédente (voir figure 4.1), les emplacements disponibles.[2]

#### 4.2.1.3 Emplacement(s) sauvegardé(s)

Après avoir tapé sur le bouton[3], une nouvelle vue apparaît (voir figure 4.3a), affichant sous forme de liste le(s) emplacement(s) mémorisé(s). On a accès à :

- l'adresse
- le code postal
- la ville
- le type de place
- le nombre de places

Plusieurs actions sont disponibles :

- **suppression** d'une place : grâce à un glissement de droite vers la gauche sur la place ou en tapant le bouton *Modifier* et tapant sur le petit symbole qui est apparu à côté de la place

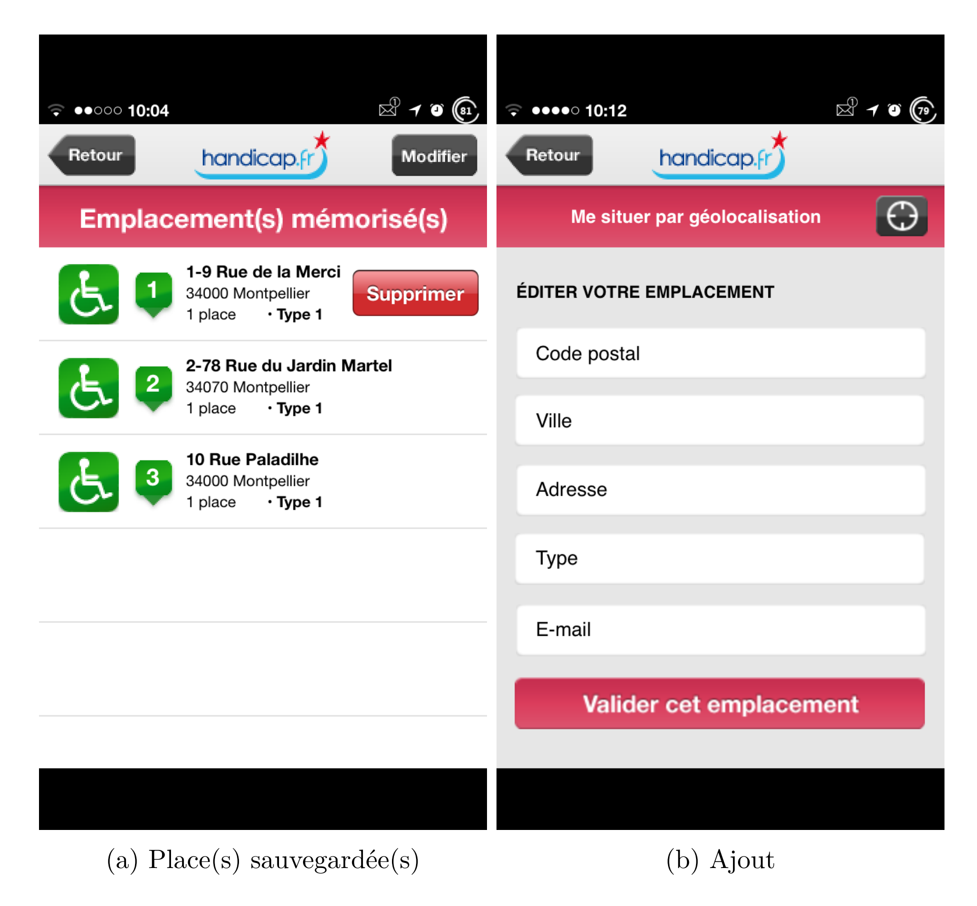
1. Apparemment, seules les grandes villes sont répertoriées.
2. Seules 51 places sont affichées au maximum.
3. Si des emplacements sont sauvegardés, une pastille apparaît sur l'icône du bouton, dénombrant les emplacements sauvegardés.

- **détails** d'une place : lorsqu'on tape sur un emplacement, une nouvelle vue apparaît, affichant la carte, centrée sur l'emplacement, affichant un bandeau d'informations en haut de la vue (ville, code postal, etc.) ainsi qu'un bouton itinéraire qui propose la même action que dans la fonctionnalité précédente.

#### 4.2.1.4 Partager un emplacement

Après avoir tapé sur le bouton, une nouvelle vue apparaît (voir figure 4.3b), invitant à renseigner différentes informations permettant la localisation de l'emplacement que l'on veut partager. On a **deux modes** pour ça : par **géolocalisation** ou par **formulaire**.
Le premier mode remplit les champs automatiquement grâce aux informations déduites par la géolocalisation (ville, adresse, etc.), il ne reste plus qu'à saisir son e-mail et le type de place. [4]
Le second mode propose de remplir un formulaire manuellement, avec les informations suivantes :

- code postal
- ville
- adresse
- e-mail
- type (place en ville, parc de stationnement, privé, administration)




(a) Place(s) sauvegardée(s) (b) Ajout

FIGURE 4.3 – Handicap.fr - Favoris et ajout

| Nom | Version | Publication | iOS mini | Compatibilité | Langue |
|---|---|---|---|---|---|
| `Handi Parking` | 2.0 | janvier 2015 | 7.0 | iPhone, iPad | Français, Anglais |

### 4.2.2 Handi Parking

Propose l'indexation des places de parking réservées aux personnes handicapées dans toutes les localités françaises.

L'application a été testée sur **iOS 8.1.2** avec un **iPhone 5S** (6.2).

L'application possède trois fonctionnalités principales :

- affichage des places proches de ma **position actuelle** grâce à la géolocalisation du téléphone (*Wi-Fi* ou *GPS*), sous la forme d'une carte avec possibilité d'obtenir l'adresse en touchant un marqueur
- recherche des places **autour d'un lieu**, sous la forme d'une carte
- soumettre un **nouvel emplacement** non répertorié

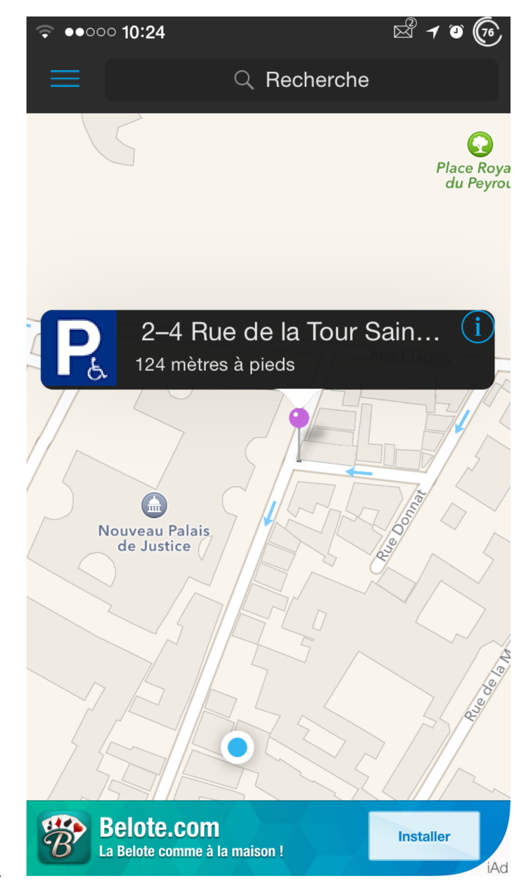



FIGURE 4.4 – Handi Parking - Place(s) proche(s) de moi

#### 4.2.2.1 Emplacement(s) proche(s) de ma position

Une carte est affichée par défaut. Pour afficher les places situées autour de ma position actuelle, il faut :

1. Sélectionner la barre de recherche, ce qui charge une nouvelle vue affichant la liste des dernières recherches par lieu ainsi qu'une ligne « *Position actuelle* »
2. Sélectionner cette ligne ce qui lance la recherche et charge la carte

Après que la recherche est terminée [5], la vue (voir figure 4.4) intègre les éléments suivants :

- notre position
- des marqueurs (les emplacements)
- les *gestures* habituels : zoom, dézoome, rotation, etc.

En effectuant un tap sur un marqueur, des informations complémentaires s'affichent :

- adresse, distance

4. Le partage par géolocalisation n'a pas marché lors des différents essais ; les champs du formulaire ne sont pas remplis automatiquement après avoir tapé sur le bouton de géolocalisation.
5. Seuls les emplacements situés dans un rayon d'environ 500m sont chargés.

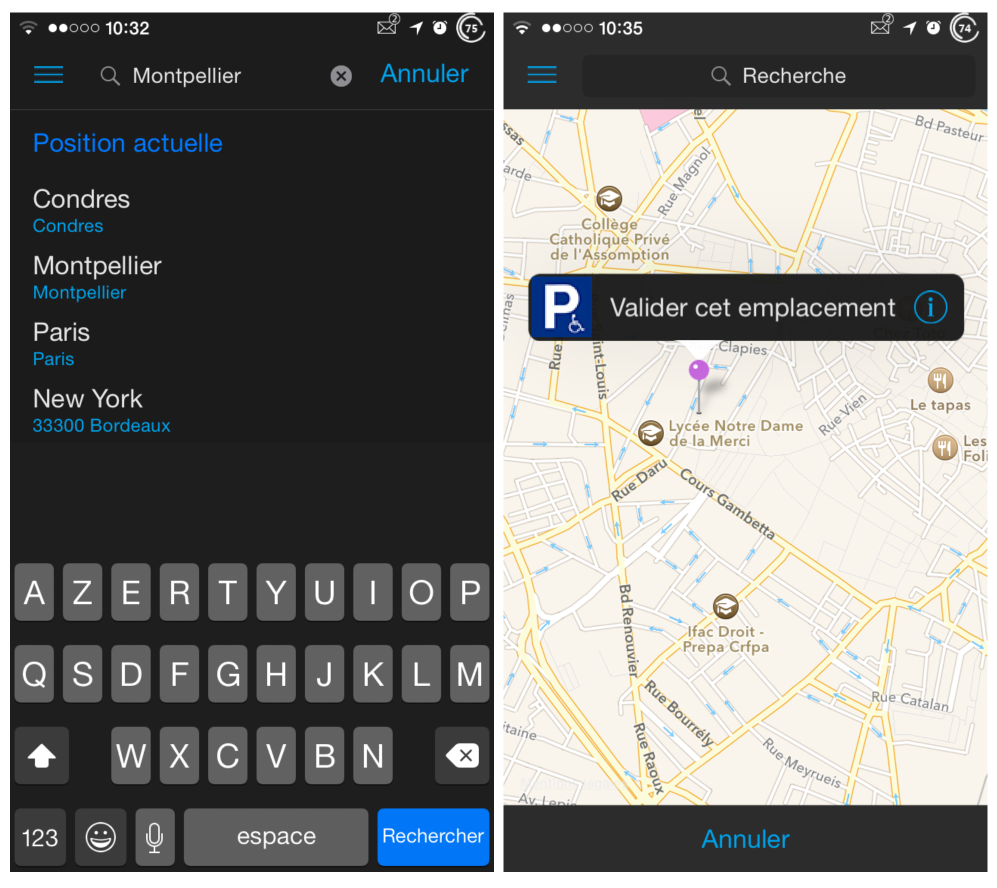
- un bouton d'informations affichant une popup pour la prise en charge de l'itinéraire par une application tierce

#### 4.2.2.2 Emplacement(s) proche(s) d'une adresse

La méthode est très similaire à la fonctionnalité. La seule différence se situe au point 2 de la recherche : il suffit de renseigner le nom de localité [6] [7] au lieu de choisir « *Position actuelle* » (voir figure 4.5a).
Mis à part ce léger changement, tout le reste est identique.

#### 4.2.2.3 Partager un emplacement

Pour partager un emplacement, il faut sélectionner le menu latéral et choisir l'unique bouton « *Ajouter un emplacement* ».
Une alerte s'affiche expliquant comment partager un emplacement : il suffit de déplacer le marqueur aux coordonnées de l'emplacement et de valider (voir figure 4.5b). La manipulation est très simple.




(a) HP - Recherche (b) HP - Valider l'ajout

FIGURE 4.5 – Handi Parking - Recherche et partage

6. La recherche n'est pas prédictive, aucune suggestion n'est proposée pour aider l'utilisateur.
7. La recherche ne fonctionne que pour un lieu situé en France, si on effectue une recherche dans un autre pays, on obtient une erreur ou alors on est redirigé sur un lieu français dont le nom correspond (de façon très approximative).

## 4.3 Fonctionnalités du système

L'utilisateur arrive sur la vue principale (voir figure 4.6) de l'application et doit avoir accès à toutes les fonctionnalités principales de l'application directement, sans avoir aucune action à effectuer.




Figure 4.6 – Vue principale

### 4.3.1 Emplacement(s) proche(s) de ma position

#### 4.3.1.1 Description

L'utilisateur doit être en mesure de rechercher les emplacements proches de sa **position actuelle**. Il n'a aucune information explicite à fournir, seulement sa position. La seule interaction est la demande de son autorisation pour l'utilisation de sa position géographique (une seule et unique fois).

#### 4.3.1.2 Spécifications détaillées

**Vérifications des conditions**

Après avoir sélectionné le bouton « *Géolocalisation* » sur l'écran principal, le système va récupérer la **position actuelle** de l'utilisateur. Plusieurs cas d'erreurs possibles :

- accès à Internet est désactivé : un message d'erreur est affiché.
- service de localisation désactivé : une erreur sera affichée aiguillant l'utilisateur pour résoudre le problème.
- l'application n'a pas le droit d'utiliser la position de l'utilisateur (première utilisation ou droit modifié dans les paramètres) : une fenêtre sera affichée pour demander l'autorisation de l'utilisateur dans le cas d'une première utilisation ou un message d'erreur dans l'autre cas (voir figure 4.7).




Figure 4.7 – Vue erreur

Cette demande est **bloquante**, si l'application n'a pas accès à la position de l'utilisateur la suite du traitement ne peut pas s'effectuer.

**Traitement des données**

Une fois la position acquise, l'application va effectuer la **requête** auprès de l'API *Overpass* (voir annexe A). Il va utiliser la position de l'utilisateur comme paramètre, afin d'obtenir les places situées autour de sa position. Par défaut, la recherche s'effectue dans un rayon de **500m**. Les données retournées sont au format *JSON*. Si la recherche retourne moins de 5 emplacements, on effectue une autre requête avec un rayon :

- d'un kilomètre
- 10 kilomètres
- 25 kilomètres

Si aucun problème ne survient, les données récupérées sont traitées pour être transformées en objet(s).
Pour donner, à chaque emplacement trouvé, une adresse approximative ainsi qu'une distance par rapport à notre position, on va utiliser l'API **Distance Matrix** de *Google Maps*.

**Affichage des résultats**

La vue est alors chargée, affichant une carte (voir figure 4.8), avec les caractéristiques suivantes :

- centrée sur la position actuelle de l'utilisateur
- chaque emplacement est représenté par un marqueur
- la carte sera dézoomée de façon à ce que tous les emplacements soient visibles sur la carte

**Actions disponibles**

Sur cette vue, plusieurs actions sont disponibles :

- en tapotant sur un marqueur, un label apparaîtra au-dessus de celui-ci
- en tapotant ce label, une nouvelle vue s'ouvrira, chargeant les informations complémentaires associées à l'emplacement (payante, itinéraire, etc.)
- l'itinéraire pourra être pris en charge par plusieurs applications tierces : *Plans*, *Google Maps*, etc. Il faut vérifier qu'elles sont bien installées avant de les afficher dans la liste des applications proposées.




FIGURE 4.8 – Vue carte

#### 4.3.1.3 Scénarios d'utilisations

**Acteurs :** l'utilisateur de l'application, le système *iOS*, *Google Maps*, *Overpass*
**Condition préalable :** système *iOS* opérationnel
**Scénario nominal :**

1. L'utilisateur choisit le bouton « *Géolocalisation* » dans l'application
2. L'application va vérifier que l'accès à Internet est activé
3. L'application va vérifier que le service de localisation est activé
4. L'application va vérifier qu'elle possède le droit d'utiliser le service de localisation
5. L'application va récupérer la position de l'utilisateur
6. L'application va envoyer une requête à *Overpass* pour récupérer la liste des emplacements
7. L'application récupère les résultats et effectue les traitements pour l'affichage de la carte (*Google Maps*)
8. L'application affiche les résultats sur la carte

**Enchaînements d'erreurs :**

**E1 :** Pas d'accès à Internet

Démarre après le point 2 du scénario nominal.

3. L'application enregistre l'échec
4. L'application affiche à l'utilisateur que l'accès à Internet est désactivé

Le scénario se termine en échec

**E2 :** Service de localisation désactivé

Démarre après le point 3 du scénario nominal.

4. L'application enregistre l'échec
5. L'application affiche à l'utilisateur que le service de localisation est désactivé

Le scénario se termine en échec

**E3 :** Droit du service de localisation

Démarre après le point 4 du scénario nominal.

5. L'application enregistre l'échec
6. L'application affiche à l'utilisateur qu'elle n'a pas l'autorisation d'utiliser le service de localisation

Le scénario se termine en échec

#### 4.3.1.4 Diagrammes de séquence

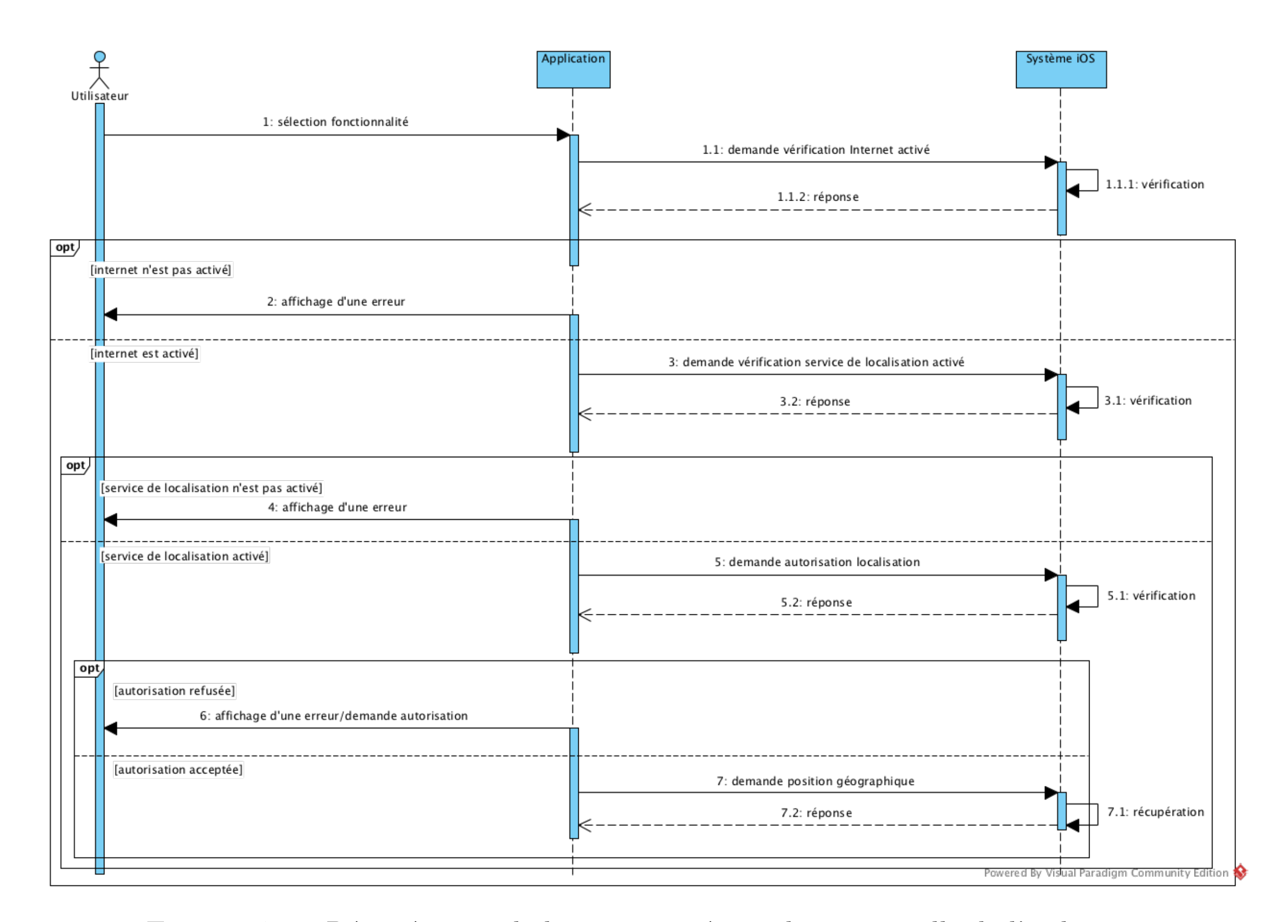



Figure 4.9 – Récupération de la position géographique actuelle de l'utilisateur

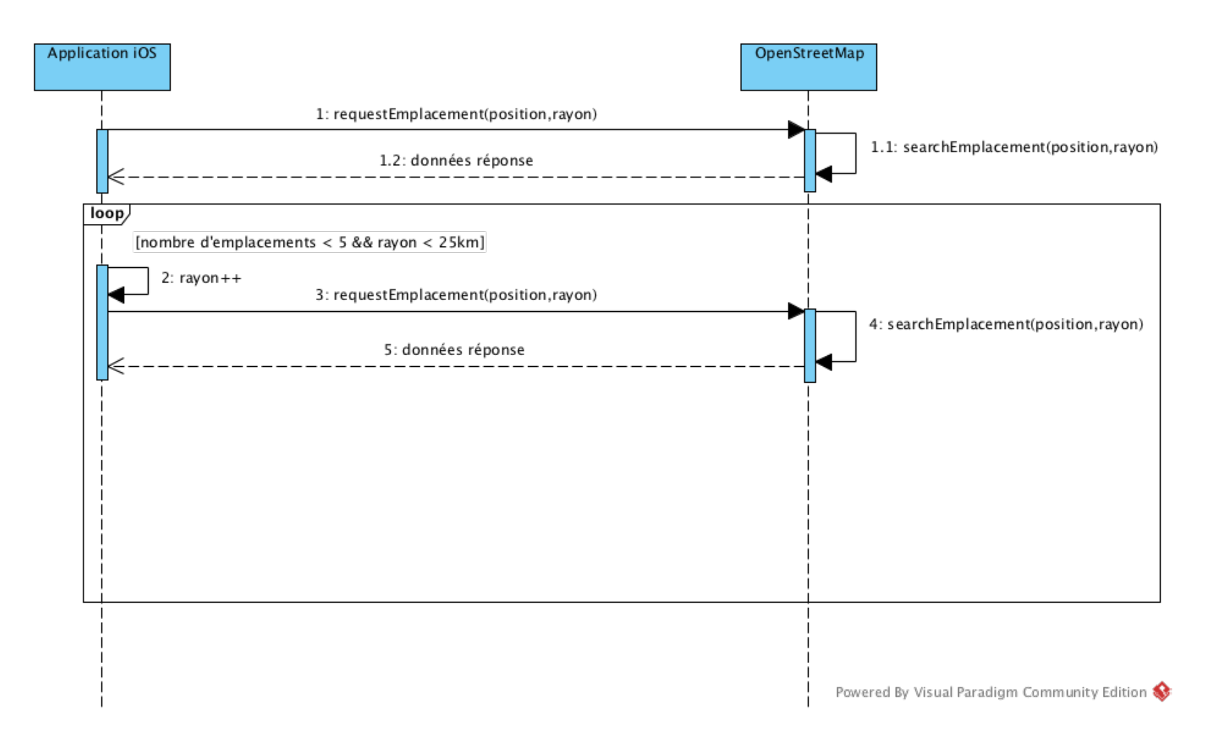



FIGURE 4.10 – Récupération des emplacements auprès d'OpenStreetMap

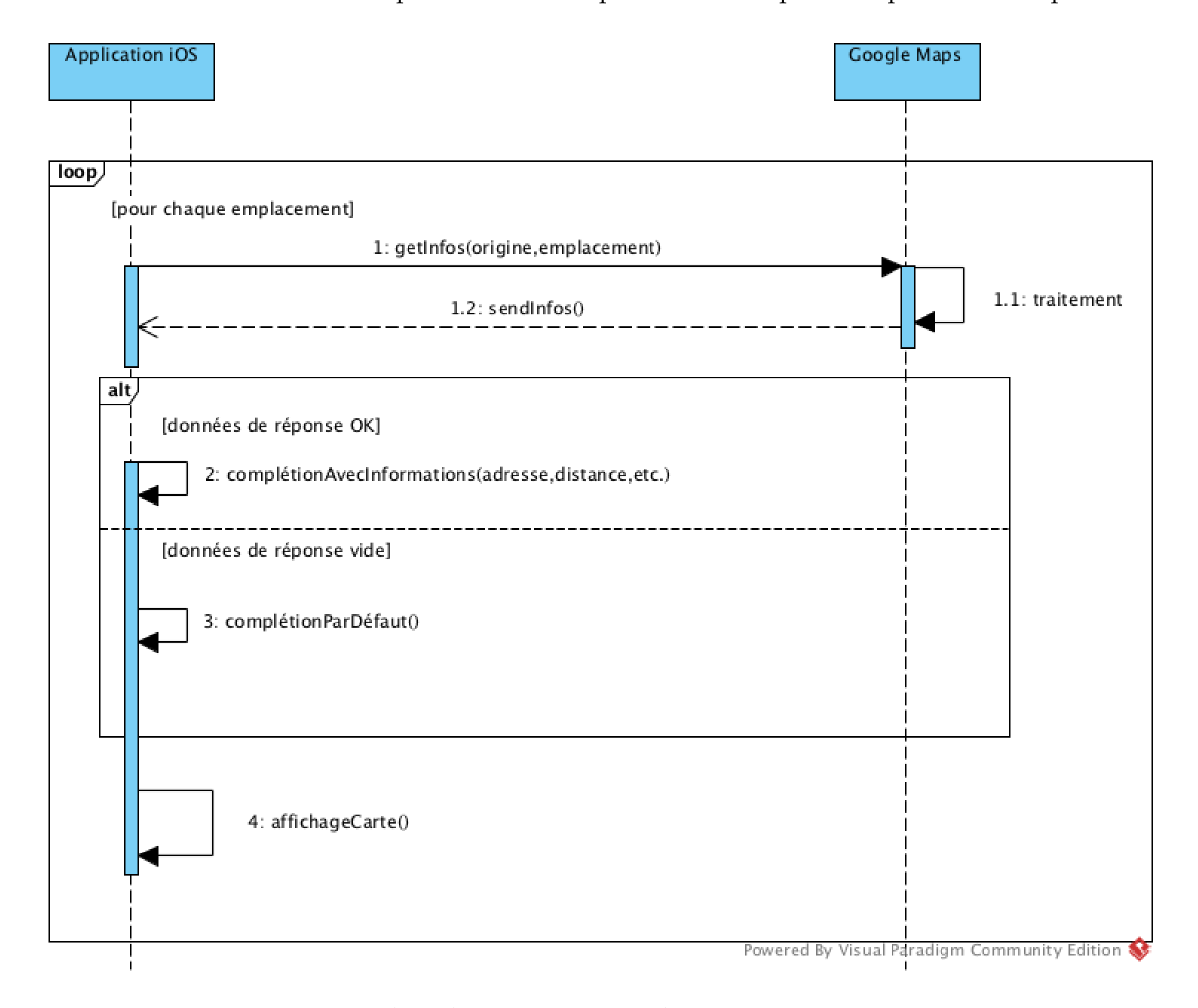



FIGURE 4.11 – Récupération des données pour chaque emplacement

### 4.3.2 Emplacement(s) proche(s) d'une adresse

#### 4.3.2.1 Description

L'utilisateur doit être en mesure de rechercher les emplacements proches d'un lieu de son choix. Pour cela, il doit saisir ce nom dans une barre de recherche et sélectionner le lieu parmi les résultats proposés. C'est la seule action que l'utilisateur doit effectuer.




FIGURE 4.12 – Vue recherche

#### 4.3.2.2 Spécifications détaillées

**Vérifications des conditions**

Après avoir sélectionné le bouton « *Rechercher* » sur l'écran principal, le système va charger une nouvelle vue (voir figure 4.12), affichant une barre de recherche. L'utilisateur est invité à saisir des caractères pour lancer la recherche. Caractéristiques de la recherche :

- ne se lance qu'après la saisie de **trois caractères ou plus**
- se lance après chaque caractère saisi
- retourne entre **0 et 5 résultats** (limite *Google Places API*)




FIGURE 4.13 – Vue résultats de recherche

Plusieurs cas d'erreurs possibles :

- accès à Internet est désactivé : un message d'erreur est affiché
- aucun résultat n'est retourné : un message est affiché

**Traitements des données**

À chaque lancement de la recherche, les données reçues sont au format *JSON*. Elles sont affichées sous forme d'une liste de lieux (voir figure 4.13). Lorsqu'un lieu est choisi, une nouvelle requête est effectuée auprès de l'API **Google Places** afin de récupérer sa latitude et sa longitude.

Une fois la latitude et la longitude récupérées, la suite du traitement est le même que celui de la fonction précédente (voir section 4.3.1).

**Affichage des résultats**

Voir section 4.3.1.

**Actions disponibles**

Voir section 4.3.1.

#### 4.3.2.3 Scénarios d'utilisations

**Acteurs :** l'utilisateur de l'application, le système *iOS*, *Google Maps*, *Overpass*
**Condition préalable :** système *iOS* opérationnel
**Scénario nominal :**

1. L'utilisateur choisit le bouton « *Rechercher* » dans l'application
2. L'application va afficher la vue de recherche
3. L'utilisateur va saisir le nom du lieu
4. L'application va vérifier que l'accès à Internet est activé
5. L'application va envoyer une requête à *Google Maps* pour récupérer la liste des lieux correspondants
6. L'application récupère les résultats et effectue les traitements pour l'affichage
7. L'application affiche les résultats sous forme de liste
8. L'utilisateur choisit un lieu
9. L'application va envoyer une requête à *Google Maps* pour récupérer les coordonnées du lieu
10. Démarre après le point 5 du scénario nominal de la fonction précédente (voir section 4.3.1)

**Enchaînements d'erreurs :**

**E1 :** Pas d'accès à Internet

Démarre après le point 4 du scénario nominal.

5. L'application enregistre l'échec
6. L'application affiche à l'utilisateur que l'accès à Internet est désactivé

Le scénario se termine en échec

**E2 :** Aucun résultat

Démarre après le point 7 du scénario nominal.

8. L'application enregistre l'échec
9. L'application affiche à l'utilisateur un message explicatif d'erreur

Le scénario se termine en échec

#### 4.3.2.4 Diagrammes de séquence

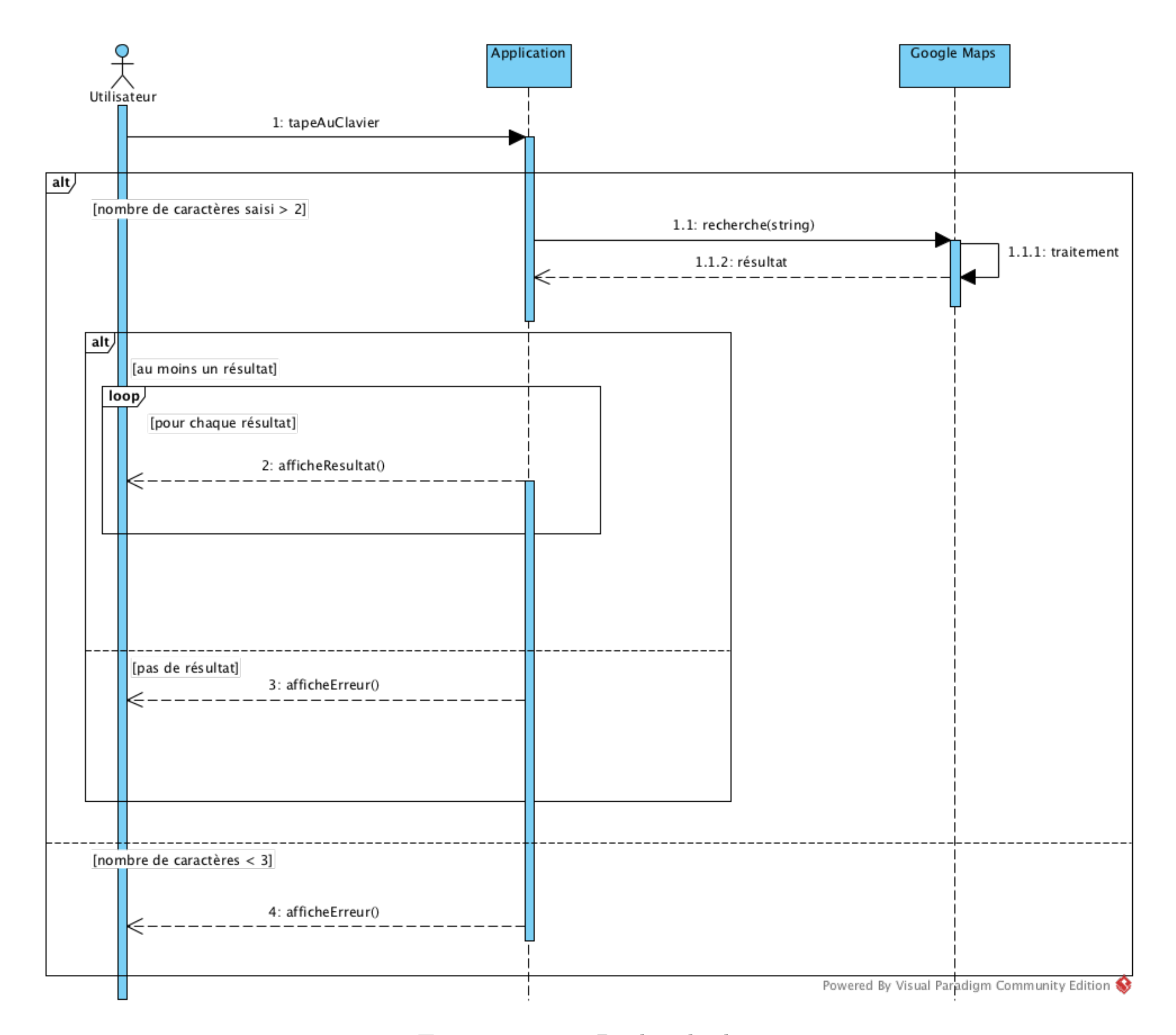



FIGURE 4.14 – Recherche lieu

## 4.4 Solutions et choix technologiques

### 4.4.1 Source de données géographiques

L'une des contraintes imposées est l'utilisation d'*open data* et notamment d'une source de données géographiques particulière, *OpenStreetMap* (OSM). OSM est un projet ayant pour but de créer une base de données géographique libre et mondiale et fournissant un accès libre aux données récoltées. Des recherches ont été effectuées sur OSM, dans le but d'en apprendre plus sur les possibilités offertes et les applications possibles à notre projet. **Cette étude est disponible en annexe A**. À l'issue de ses recherches, le format de récupération des données depuis OSM a été choisi : le *JSON*.

### 4.4.2 Le rendu des données cartographiques

Un des choix les plus importants a été celui du support pour le rendu des données. Le choix s'est porté sur *Google Maps* pour plusieurs raisons : intégration facile, fonctionnalités fournies par le SDK très importantes, carte avec un contenu très riche et très grande exactitude des données. En plus du SDK de *Google Maps*, deux autres API liées à *Google Maps* ont été utilisées :

- **Places API** : effectuer une recherche de lieu par mot clé et obtenir toutes les informations sur un lieu donné
- **Distance Matrix API** : définir l'adresse de deux points, ainsi que la distance et la durée du trajet entre les deux-points

## 4.5 Problématiques & évolutions futures

### 4.5.1 Problématiques

#### 4.5.1.1 OpenStreetMap

Ce projet étant un projet libre, de nombreux services sont publics. Cela implique que ceux-ci peuvent être arrêtés à tout moment ou que leur qualité puisse varier lors de forte charge. En revanche, cela implique aussi un certain niveau de sécurité quant à l'ajout, la modification ou la suppression d'informations sur les serveurs d'OSM. Il faut posséder un compte utilisateur pour effectuer de telles actions. Il faudrait ajouter un système de connexion en utilisant l'API REST d'édition d'OSM. C'est un système assez lourd à mettre en place, étant à gérer de multiples problèmes tels que la synchronisation, la multiédition ou les conflits. C'est une des évolutions futures envisageables.

### 4.5.2 Évolutions futures

Le projet est loin d'être abouti, seules les fonctionnalités les plus essentielles ont été choisies. Voici une liste non exhaustive des évolutions futures à implémenter :

- **emplacements sous forme de liste** : en plus d'avoir les emplacements représentés sous forme cartographique, il serait intéressant d'avoir une liste des emplacements triée par ordre croissant de distance
- **favoris** : il serait intéressant d'avoir la possibilité de sauvegarder un emplacement afin de le retrouver plus facilement ou si l'on s'y rend souvent par exemple
- **signaler/supprimer un emplacement** : avoir la possibilité de supprimer une place implique qu'il faut un compte OSM ; signaler une place et attendre son approbation ou son rejet par la communauté serait un moyen plus efficace d'éviter les données frauduleuses
- **proposer/ajouter un emplacement** : même chose que pour signaler/supprimer
- **compléter/modifier un emplacement** : même chose que pour signaler/supprimer
- **filtrage grâce aux tags** : il serait intéressant d'utiliser les nombreux tags que fournis OSM pour filtrer les emplacements trouvés ; exemple : le tag *fee=no/yes* pour savoir si le stationnement est gratuit/payant
- **informations complémentaires** : il serait intéressant d'utiliser les nombreux tags que fournis OSM et qui sont associés à un emplacement pour afficher toutes les informations disponibles
- **StreetView** : grâce au SDK de *Google Maps*, il serait intéressant d'ajouter la possibilité d'une StreetView de l'emplacement

Chapitre 5

# Rapport technique

Cette partie technique présente dans un premier temps les technologies utilisées pour développer l'application ainsi que le fonctionnement général de celle-ci. Dans un second temps, un examen des résultats obtenus par rapport aux résultats attendus est effectué suivi d'une revue des perspectives énoncées dans les spécifications.

## 5.1 Choix technologiques

### 5.1.1 Outils & langages

#### 5.1.1.1 Outils

Pour le développement de ce projet, plusieurs outils ont été utilisés, dont certains indispensables :

- **un Mac** : en effet le développement d'applications OS X ou iOS se fait presque exclusivement sur OS X. Certes, il est également possible de développer sous Windows ou Linux, mais ce n'est pas la voie la plus simple.
- **Xcode** : un environnement de développement pour OS X et iOS. Il contient le compilateur LLVM, les derniers SDK pour iOS et OS X, *iOS Simulator* et dans ses dernières versions *Swift*.
- **iOS Simulator** : simulateur iOS inclus dans Xcode permettant de tester l'application sur différents appareils et versions d'iOS et incluant de nombreux outils de débug.
- **appareil sous iOS** : un iPhone en l'occurrence pour tester l'application sur un appareil physique en plus du simulateur.
- **Network Link Conditioner** : un utilitaire qui permet aux appareils Mac et iOS de simuler des environnements réseaux défavorables.
- **Subversion** : ou SVN, logiciel de gestion de versions de fichiers

#### 5.1.1.2 Swift

Un langage était imposé pour ce projet, Swift, nouveau langage développé par Apple et destiné pour le développement d'applications iOS et OS X. Présenté en juin 2014, il est prévu pour coexister avec l'Objective-C, l'actuel et unique langage permettant de développer des applications iOS et OS X.

### 5.1.2 SDKs

#### 5.1.2.1 iOS Software Development Kit

Apple publie généralement un nouveau SDK à chaque mise à jour majeure (exemple : iOS 8.0) et mineure (exemple : iOS 8.1) d'iOS. Au début de la phase de développement, début mars 2015, la version SDK était 8.1 c'est donc cette version qui a été utilisée comme base. Courant mars 2015, Apple a publié iOS 8.2 et avec elle le nouveau SDK 8.2. Ainsi une première migration a été effectuée vers ce nouveau SDK.
Début avril, Apple a publié iOS 8.3 et le SDK correspondant. En même temps que ce SDK, Apple a mis à jour Swift passant de la version 1.1 à la 1.2. Une migration a donc dû être effectuée pour se conformer aux nombreux changements de syntaxe. C'est cette version qui sert comme SDK de base pour l'application. Également en même temps que ce SDK, Apple a publié la première version bêta d'iOS 8.4. Des essais de l'application ont été effectués sur cette nouvelle version et celle-ci semble parfaitement fonctionnelle.

L'iOS SDK est divisé en plusieurs couches :

- **Cocoa Touch** : framework dérivé de *Cocoa* gérant l'interface utilisateur, l'apparence de l'application et étant organisé selon le design pattern MVC
- **Media** : cette couche contient toutes les technologies multimédia : images, audio et vidéo
- **Core Services** : contient les services systèmes fondamentaux comme la géolocalisation ou *iCloud*
- **Core OS** : contient le système principal (gestion processeurs, mémoire, I/O, énergie, etc.) ainsi que les fonctionnalités bas niveau : Bluetooth, sécurité, réseau, etc.

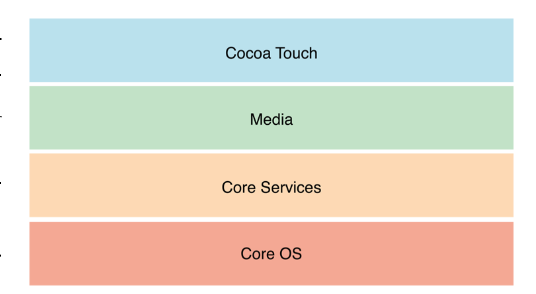



FIGURE 5.1 – Architecture iOS

Pour pouvoir télécharger le SDK (ou plutôt Xcode qui contient le SDK), il faut posséder un compte développeur Apple (gratuit).

#### 5.1.2.2 Google Maps SDK for iOS

Jusqu'à iOS 5, *Google Maps* faisait partie intégrante d'iOS et était le moteur de carte utilisé par les appareils iOS. Avec la sortie d'iOS 6 en 2012, Apple a remplacé *Google Maps* avec son propre moteur de carte, *Apple Maps*.
Bien qu'avec le temps *Apple Maps* ait pris une bonne part du marché, il lui reste encore beaucoup de chemin à parcourir avant d'arriver au même niveau que *Google Maps*, qui reste encore aujourd'hui le moteur de carte favori des utilisateurs iOS.

La dernière version du SDK est la 1.9.2, datant de février 2015 et c'est celle utilisée dans l'application. Swift étant un langage assez jeune, le SDK de *Google Maps* est encore écrit en Objective-C. Comme l'application est entièrement développée en Swift, il a fallu la lier aux librairies afin de pouvoir utiliser les fonctionnalités du SDK dans l'application.

Pour fonctionner, *Google Maps SDK for iOS* a besoin d'une clé API iOS. Pour en créer une, il suffit de posséder un compte Google (gratuit) et de se rendre dans l'espace développeur de Google. Pour créer la clé, on a besoin du *bundle identifier* (basé sur un système de reverse DNS) de l'application, afin que celle-ci ne soit valide uniquement que pour cette application.

Avec *Google Maps SDK for iOS*, on peut ajouter des cartes *Google Maps* à notre application. Le SDK gère automatiquement les accès aux serveurs de *Google Maps*, l'affichage et les gestures de l'utilisateur. On peut ajouter des objets sur la carte comme des marqueurs, objets qui peuvent contenir des informations supplémentaires, comme une fenêtre pour les marqueurs. En revanche, le SDK ne permet pas de faire des requêtes plus complexes que de l'affichage. On doit passer par un autre service, l'API Google Places.

### 5.1.3 APIs

#### 5.1.3.1 Google Places API

L'API Google Places est un service Web qui retourne des informations sur un lieu — défini dans cette API comme établissements, lieux géographiques, ou points d'intérêt — en utilisant des requêtes HTTP. Il n'y a pas pour le moment d'API native pour iOS. [1]

Pour fonctionner, *Google Places API* a besoin d'une clé API serveur. Pour en créer une, il suffit de posséder un compte Google (gratuit) et de se rendre dans l'espace développeur de Google. Plusieurs types de requêtes sont possibles, mais seulement deux ont été utilisées dans l'application :

- **Place Details** : retourne des informations détaillées à propos d'un lieu
- **Place Autocomplete** : retourne une liste de lieux (prédictions) correspondant à la chaîne de caractères passée en paramètres

Les deux services sont utilisés de façon complémentaire : **Place Autocomplete** est utilisé pour fournir une liste de lieux prédictifs lors de la recherche d'un lieu par l'utilisateur. Ce service renvoie le nom du lieu ainsi qu'un identifiant unique généré par Google. **Place Details** utilise cet identifiant unique pour récupéré toutes les informations sur ce lieu (coordonnées géographiques, adresse, etc).

#### 5.1.3.2 Overpass API

Voir annexe A.

### 5.1.4 Frameworks

#### 5.1.4.1 MapKit

*MapKit* est un framework d'Apple, qui permet d'incorporer des cartes dans des applications iOS. Elle fournit également des fonctionnalités supplémentaires comme le calcul d'un itinéraire entre deux lieux ou la recherche de lieu basé sur des mots clés. *MapKit* est inclus dans l'iOS SDK, il suffit simplement de l'importer pour pouvoir l'utiliser.

Dans notre application, *MapKit* sert à calculer la distance et le temps de trajet estimés entre deux points, entre la position actuelle de l'utilisateur et l'emplacement de la place choisie.

#### 5.1.4.2 Autres frameworks

Bien que Swift soit un langage jeune, il y a déjà une grande prolifération de projets open source construits avec Swift. Quelques-uns de ses projets ont été utilisés dans l'application, car très intéressants et permettant de gagner un temps de développement considérable. La plupart

1. Fin mars 2015, Google a lancé *Google Places API for iOS* en version bêta, ce qui pourrait permettre une possible intégration native dans l'application.

de ses projets ont été modifiés ou vus leurs fonctionnalités augmentées pour répondre aux besoins de l'application.

- **Alamofire** : *Alamofire* est un framework facilitant les appels réseau depuis une application iOS. Il s'appuie sur *NSURLSession* et *Foundation URL Loading System*, deux services réseau d'Apple, pour fournir des services et méthodes réseaux de haut niveau.
  *Alamofire* gère les appels HTTP, l'authentification, les flux, l'upload/download avec état de progression, etc.
  Les appels réseau sont effectués de façon asynchrone (voir section 5.2.3). Dans l'application, *Alamofire* est utilisé pour effectuer les requêtes aux API Web, Google Place et *Overpass*.
- **SwiftyJSON** : JavaScript Object Notation ou JSON est une manière courante de transmettre des données entre différents services Web. Swift est très strict concernant les types. Chaque objet JSON doit être vérifié via optional binding (ou liaison optionnelle) avant de pouvoir être utilisé. Cela rend le code sûr, mais dès qu'on commence à gérer des objets JSON complexes, le code devient moche et compliqué.*SwiftyJSON* fournit une façon beaucoup plus simple de gérer les objets JSON sans avoir à utiliser de liaison optionnelle.
- **Alamofire-SwiftyJSON** : *Alamofire-SwiftyJSON* est une extension d'*Alamofire*, permettant de sérialiser les réponses en JSON grâce à *SwiftyJSON*. Dans l'application, *Alamofire-SwiftyJSON* est utilisé pour parser les données récupérées par les requêtes *Alamofire*.
- **SwiftSpinner** : *SwiftSpinner* est un indicateur d'activité, utilisant un flou dynamique et la transparence pour recouvrir la vue actuelle. Il affiche un indicateur d'activité circulaire aussi appelé « spinner ». Dans l'application, *SwiftSpinner* est utilisé lors de la recherche d'emplacements de places.
- **SCLAlertView** : *SCLAlertView* est un framework affichant des vues d'alertes personnalisées et animées. Dans l'application, *SCLAlertView* est utilisé en remplacement des vues d'alertes standards.
- **IJReachability** : *IJReachability* est une classe permettant de vérifier la disponibilité de la connexion Internet, que ce soit en Wi-Fi ou via les données cellulaires.
- **RAMAnimatedTabBar** : *RAMAnimatedTabBar* est un module permettant d'ajouter un animation aux onglets de la barre d'onglets.
- **DTIActivityIndicator** : *DTIActivityIndicator* est un indicateur d'activité proposant plusieurs animations. Dans l'application, *DTIActivityIndicator* est utilisé pour signaler l'activité lors du chargement d'informations d'un marqueur par exemple.
- **LTMorphingLabel** : *LTMorphingLabel* est un framework proposant d'ajouter des effets de transformations sur les textes. C'est une sous-classe de *UILabel*.

## 5.2 Conception

### 5.2.1 Architecture

Le modèle Modèle Vue Contrôleur (MVC) est une des pierres angulaires de *Cocoa*, l'API d'Apple pour développer des applications, et c'est sûrement le design pattern le plus utilisé de tous. Il organise les objets en fonction de leur rôle dans l'application et encourage une séparation claire du code.

Les trois rôles sont :

- le **modèle** : l'objet qui gère les données de l'application et définit les méthodes qui les manipulent.

- la **vue** : l'objet en charge de la représentation visuelle du modèle et des contrôles avec lesquels l'utilisateur peut interagir ; généralement, ce sont tous des objets dérivant de la classe *UIView*
- le **contrôleur** : c'est le médiateur, celui qui coordonne tout le travail. Il accède au données grâce au modèle et les affichent grâce à la vue, écoute les évènements et manipule si besoin les données.

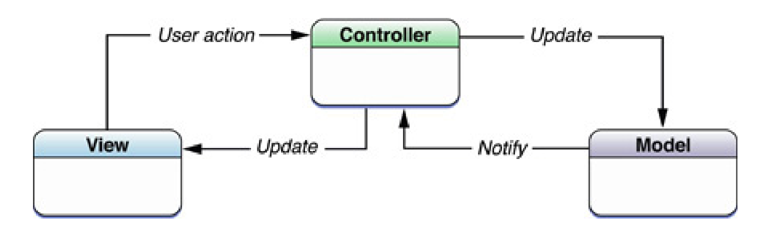



FIGURE 5.2 – Architecture de l'application

En plus, de cette architecture principale, une autre structure est présente, **DataProvider**. Cette structure peut-être comparée à un routeur, elle va construire l'URL (avec ses paramètres) à utiliser pour les appels aux API non natives. Elle est composée de simples énumérations se conformant au protocole *URLRequestConvertible*, protocole défini dans *Alamofire*. C'est cette structure qui contient, en dur, les URL de base des API ainsi que les clés d'API. Ainsi chaque énumération a plusieurs points de sortie, chacun correspondant à une URL différente.

## 5.2.2 Interface utilisateur

### 5.2.2.1 Auto Layout

*Auto Layout* est un système qui permet de gérer la mise en page de l'interface utilisateur de l'application en créant une description mathématique des relations entre les éléments, en termes de contraintes.

Ainsi, toutes les vues de l'application s'adaptent automatiquement à l'écran, que ce soit en fonction de la taille de l'écran de l'appareil, son orientation et même de la langue.

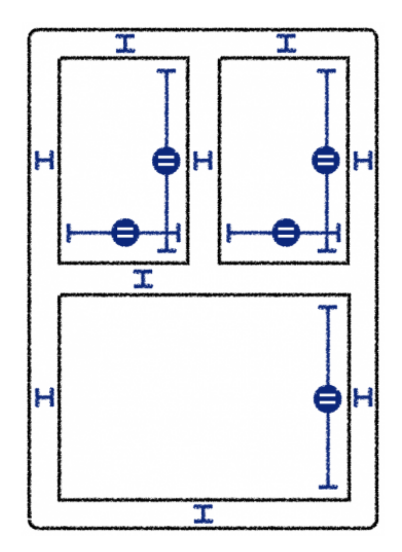


FIGURE 5.3 – Contraintes *Auto Layout*

### 5.2.2.2 Menu

Un « menu hamburger » ou « sidebar » est un panneau coulissant qui sort de l'écran lorsqu'on swipe son coin gauche ou qu'on tape le bouton représentant trois barres. C'est une méthode de design assez connue pour ajouter (ou cacher) des fonctionnalités ailleurs que sur l'écran principal. Ce type de menu est très largement utilisé dans le monde des applications mobiles pour ajouter un menu sur le côté.
Bien que l'on gagne de la place sur l'écran avec ce menu, Apple [5] (et beaucoup d'autres) déconseille son utilisation, et ce pour une bonne raison : il ne respecte pas les trois points clés d'un système de navigation intuitif :

- où suis-je ?
- où puis-je aller ?
- que vais-je trouver là-bas ?

À la place, d'un menu hamburger il a été choisi d'utiliser un menu d'onglets (ou Tab Bar).

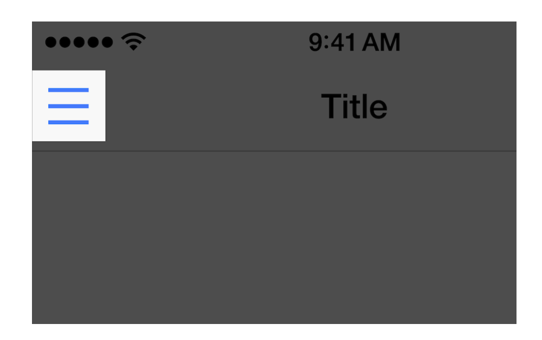



FIGURE 5.4 – Menu hamburger

### 5.2.3 Design patterns

Les design patterns sont des solutions réutilisables pour des problèmes récurrents dans la conception de logiciels. Ce sont des modèles conçus pour aider à écrire du code qui n'est pas facile à comprendre et à réutiliser. Ils permettent également de modifier ou changer des composants dans le code sans trop de difficulté.
Dans l'application, plusieurs design patterns ont été combinés.

#### 5.2.3.1 Singleton

Le design pattern *singleton* s'assure qu'une seule et unique instance de la classe est créée et qu'il n'y a qu'un seul point d'accès global à cette instance.
Apple utilise beaucoup cette approche : par exemple, une application ne possède qu'une seule fenêtre. Dans notre application, ce design pattern est utilisé par le framework *SwiftSpinner* : il n'y a une seule instance du spinner qui est utilisée dans toute l'application.

#### 5.2.3.2 Decorator

Le design pattern *decorator* ajoute dynamiquement des comportements et responsabilités à un objet sans modifier son code. En Swift, il y a deux implémentations majeures de ce design pattern : *extension* et *délégation*.

**Extension**

Créer des *extensions* est un mécanisme puissant qui permet d'ajouter des fonctionnalités à une classe, structure ou énumération existante sans avoir à créer de sous-classe. Une des caractéristiques intéressantes de ce design pattern, c'est que l'on peut étendre du code auquel on n'a pas accès, comme les classes de l'API *Cocoa*. Les méthodes sont ajoutées à la compilation et peuvent être utilisées comme des méthodes normales.

Dans notre application, plusieurs frameworks utilisent ce design pattern : *Alamofire*, *SwiftyJSON*, etc. Pour les besoins de l'application, la classe *UIView* a été étendue.

**Delegate**

La *délégation* est un design pattern qui existe sur beaucoup de plates-formes, mais Apple l'a adoptée et l'utilise énormément dans ses APIs. C'est un design pattern qui transmet la responsabilité d'une classe à une autre.

En pratique, la *délégation* est très souvent utilisée comme un moyen pour une classe, de communiquer avec un autre classe. Plus particulièrement, cela permet à une instance de signaler

à une autre instance (son delegate) que quelque chose va se passer ou s'est passé.

Dans notre application, on peut citer plusieurs implémentations de ce procédé : lorsqu'un marqueur est touché ou que le bouton de fermeture de la vue de recherche est touché, le contrôleur est notifié et il peut effectuer les actions nécessaires.

Une autre utilisation commune de la *délégation* est de fournir au delegate l'opportunité de modifier certaines propriétés internes de l'instance qui délègue.

#### 5.2.3.3 Observer

Dans le design pattern *observer*, un objet notifie d'autres objets d'un changement d'état. Les objets n'ont pas besoin de se connaître entre eux. Ce design pattern est surtout utilisé pour notifier certains objets quand une propriété est modifiée. L'implémentation normale demande que l'observateur enregistre son intérêt pour l'état d'un autre objet. Quand l'état change, tous les objets observateurs sont notifiés du changement.

*Cocoa* implémente ce pattern de deux façons : *notifications* et *KVO*. Dans l'application seul *KVO* a été utilisé, notifications ne présentant pas d'intérêt.

**Key-Value Observing (KVO)**

Avec *KVO*, un objet peut demander à être notifié à chaque changement de valeur d'une propriété spécifique, que ce soit la sienne ou celle d'un autre objet.
Dans notre application par exemple, on demande à *KVO* d'être notifié dès qu'un marqueur de la carte est sélectionné ou désélectionné.

#### 5.2.3.4 Proactor

*Proactor* est un design pattern pour la gestion des évènements dans lesquels une partie de l'activité est asynchrone. Un *completion handler* est appelé lorsque la partie asynchrone est terminée.
*Grand Central Dispatch* (GCD) est une librairie d'Apple, implémentant ce design pattern. Il permet le support d'une exécution concurrente de code sur un appareil multicoeur. GCD peut améliorer la réactivité de l'application en exécutant des tâches longues et coûteuses en arrière-plan et ainsi ne bloquant pas l'exécution.

Il est d'usage d'éviter l'exécution de tâches lourdes sur le thread principal et à raison : le thread principal gère l'interface utilisateur et les interactions avec celui-ci. Exécuter une tâche trop longue aurait pour effet de bloquer le main thread et ainsi freeze l'interface utilisateur.

*Alamofire* utilise GCD pour effectuer ses requêtes en arrière-plan : au lieu de bloquer l'exécution et d'attendre la réponse du serveur, un *callback* (*completion handler*) est spécifié pour gérer la réponse lorsqu'elle est reçue. Ainsi l'exécution continue sans attendre la réponse du serveur. Le résultat de la requête est seulement disponible dans le *completion handler*. Toute exécution dépendante de la réponse ou des données reçues du serveur doit être effectuée dans le handler. Ici le *completion handler* est une trailing closure ou « closure expression ».

Une closure est une portion de code qui capture et garde les références de chaque variable et constante dans le contexte dans lequel elles ont été définies, même après que l'exécution soit passée à un autre bloc. Ce concept est connu sous le nom de « fermeture », d'où le nom closure. Une trailing closure est une forme particulière de closure, syntaxiquement plus courte.

## 5.3 Résultat du développement

L'application a été développée entièrement en Swift. Cette section décrit le résultat du développement qui a été effectué. Les principales fonctionnalités décrites ci-dessous sont les fonctionnalités les plus importantes de notre projet.
Nous détaillions uniquement 6 fonctionnalités, ainsi que les aspects fonctionnels les plus importants de notre application.

### 5.3.1 Recherche de places à proximité

La recherche par géolocalisation est une des fonctionnalités principales de notre projet, elle permet de rechercher les places de parking situées à proximité de notre position. C'est cette fonctionnalité qui est affichée par défaut au lancement de l'application. C'est le contrôleur **GeoViewController** qui gère cette vue.

#### 5.3.1.1 Cas d'erreurs

Quatre cas d'erreurs sont traités :
- pas d'accès Internet
- service de localisation désactivé
- pas le droit d'utiliser la localisation
- temps de réponse trop important (supérieur à 10 secs)

Chacune de ses erreurs est bloquante et génère un message d'erreur permettant à l'utilisateur de résoudre le problème.

#### 5.3.1.2 Cas nominal

Tous les services sont fonctionnels et l'utilisateur lance la recherche, par défaut dans un rayon de 500m. Si moins de 10 emplacements sont trouvés, on relance une recherche en augmentant le rayon, jusqu'à trouver un nombre d'emplacements suffisants ou avoir un rayon de recherche égal à 25km. Lorsque la recherche se termine (succès ou non), on traite les résultats s'il y en a pour n'avoir que les emplacements les plus proches et on les affiche sur la carte. S'il n'y en a pas, un message d'erreur est affiché.
Si un serveur ne répond pas (timeout) après 10 secondes, on vérifie si un autre est disponible et on l'utilise, sinon on affiche une erreur.
Pendant cette recherche, un spinner est affiché à l'écran bloquant toute interaction. On peut stopper la recherche et revenir sur la vue de départ.

#### 5.3.1.3 Fonctions importantes utilisées

- **launchAction()** : vérifie que tous les services sont fonctionnels
- **launchRecherche()** : remet les paramètres de recherche à zéro et initie la recherche
- **getEmplacements(*coordinate: CLLocationCoordinate2D, radius: SearchRadius*)** : effectue la requête asynchrone auprès d'*Overpass*, traite la réponse et créer les objets emplacements
- **searchResultsController()** : vérifie les résultats de recherche, en lançant la préparation du rendu (au moins 10 emplacements ont été trouvés ou rayon supérieur à 25km) ou une nouvelle recherche
- **sortAndFilterNearestPlace()** : tri les emplacements trouvés par ordre décroissant selon la distance à vol d'oiseau de la position de l'utilisateur

- **createMarkersAndBoundsToDisplay()** : création des marqueurs et affichage sur la carte ; calcul des bornes de la carte pour ajuster le zoom de la caméra afin que tous les marqueurs soient affichés sur la carte
- **switchServer()** : change le serveur *Overpass* sur lequel on effectue nos appels
- **didTapCloseButton()** : arrête la recherche en cours

### 5.3.2 Informations sur un emplacement

Après avoir terminé la recherche avec succès, on peut afficher plus d'informations sur un emplacement en tapant son marqueur. C'est également le contrôleur **GeoViewController** qui gère cette fonctionnalité.

#### 5.3.2.1 Cas d'erreurs

Quatre cas d'erreurs sont traités :

- pas d'accès Internet
- service de localisation désactivé
- pas le droit d'utiliser la localisation
- temps de réponse trop important

Chacune de ses erreurs est bloquante et génère un message d'erreur permettant à l'utilisateur de résoudre le problème.

#### 5.3.2.2 Cas nominal

Tous les services sont fonctionnels, la recherche a retourné au moins un résultat, qui est affiché sur la carte sous forme de marqueur. L'utilisateur sélectionne un marqueur pour obtenir plus d'informations sur celui-ci. Une fenêtre s'affiche au-dessus du marqueur, ce qui lance la récupération de l'adresse approximative (via Google Maps SDK) et de la distance et durée de parcours estimée (via *MapKit*) [2]. Lorsque ces informations sont récupérées, elles sont affichées dans la fenêtre. Si le délai de récupération est dépassé, la fenêtre s'affiche quand même, mais sans ses informations, uniquement les données récupérées sur *Overpass*.

#### 5.3.2.3 Fonctions importantes utilisées

- **mapView(*mapView: GMSMapView !, didTapMarker marker: GMSMarker !*) -> Bool** : lance la récupération des informations complémentaires ; on retourne un booléen (faux) pour que la fenêtre soit affichée (comportement de Google Maps SDK)
- **mapView(*mapView: GMSMapView !, markerInfoWindow marker: GMSMarker !*) -> UIView !** : charge notre fenêtre personnalisée (init) ; si les informations ont été récupérées, elles sont ajoutées dans la fenêtre, sinon on affiche un message d'attente à la place ; on retourne une *UIView* (notre fenêtre) pour l'afficher
- **loadInfoWindow()** : ajoute un spinner pendant la récupération des informations afin de signaler l'activité à l'utilisateur
- **reverseGeocodeCoordinate(*place: PlaceMarker*)** : effectue la requête asynchrone auprès de Google (Google Maps SDK) pour récupérer de l'adresse approximative de la place sélectionnée

2. Au lieu de récupérer ses informations pour tous les marqueurs lors de la recherche, cette récupération ne s'effectue que si on souhaite afficher les informations du marqueur. Cela permet de gagner du temps lors de la recherche et d'économiser de la bande passante, car l'accès internet n'est pas illimité sur tous les forfaits téléphoniques.

- **getExpectedDistanceAndTravelTime(*place: PlaceMarker*)** : effectue la requête asynchrone auprès d'Apple (*MapKit*) pour récupérer la distance et le temps de parcours estimé entre la place et la position actuelle de l'utilisateur (voir annexe C)

### 5.3.3 StreetView et itinéraire

Après avoir sélectionné un marqueur et récupéré ses informations ou non, deux nouveaux boutons sont disponibles.

#### 5.3.3.1 StreetView

On peut afficher un panorama *StreetView* du marqueur sélectionné. C'est le contrôleur **StreetViewController** qui gère cette fonctionnalité.

**Cas d'erreurs**

Trois cas d'erreurs sont traités :
- pas d'accès Internet
- temps de réponse trop important
- *StreetView* non disponible pour ce lieu

Chacune de ses erreurs est bloquante et génère un message d'erreur permettant à l'utilisateur de résoudre le problème.

**Cas nominal**

Tous les services sont fonctionnels et la récupération des informations est terminée (succès ou non). L'utilisateur a sélectionné le bouton « *StreetView* ». Une nouvelle apparaît, par-dessus l'actuelle et charge *StreetView*. L'utilisateur peut se déplacer à l'intérieur comme dans un panorama *StreetView* lambda.

**Fonctions importantes utilisées**

- **viewDidLoad()** : instanciation de la vue *StreetView* ; initialisation du service, création de la caméra et sa position et enfin du panorama
- **generateLoadingView()** : ajoute un spinner pendant la récupération des informations de la vue
- **closeButton()** : ferme la vue *StreetView* et revient sur la vue précédente

#### 5.3.3.2 Itinéraire

On peut afficher une liste des applications capables de prendre en charge l'itinéraire jusqu'à cet emplacement. C'est le contrôleur **GeoViewController** qui gère cette fonctionnalité.

**Cas d'erreurs**

Aucun cas d'erreur n'est traité : en effet, on a toujours au moins une application, *Plans*, installée par défaut sur iOS 8.

**Cas nominal**

Tous les services sont fonctionnels et la récupération des informations est terminée. L'utilisateur a sélectionné le bouton « Itinéraire ». Cela a pour action de lister les applications permettant de prendre en charge l'itinéraire : en comparant une liste pré établie dans l'application avec la liste des applications installées. Une fois la liste établie, elle est affichée à l'utilisateur, qui peut en sélectionner une, ce qui la lance en mode navigation.

**Fonctions importantes utilisées**

- **actionSheet(*sheet: UIActionSheet, clickedButtonAtIndex buttonIndex: Int*)** : ouvre l'application sélectionné avec les paramètres de navigation générés
- **getListOfInstalledMapsApps()** : récupère la liste des applications de navigation installées
- **generateURLScheme(*appName:String, location: CLLocationCoordinate2D, marker: PlaceMarker*) -> NSString** : génère l'URL scheme complète contenant les paramètres permettant d'ouvrir l'application en mode navigation
- **generateActionSheet()** : génère la liste (vue) des applications installées

### 5.3.4 Recherche d'un lieu

La recherche d'un lieu est une des fonctionnalités principales de notre projet, elle la possibilité de rechercher et choisir un lieu grâce à son nom. Ce sont les contrôleurs **SearchViewController** et **BaseTableView** qui gèrent cette vue.

#### 5.3.4.1 Cas d'erreurs

Deux cas d'erreurs sont traités :
- pas d'accès Internet
- temps de réponse trop important (supérieur à 10 secs)

Chacune de ses erreurs est bloquante et génère un message d'erreur permettant à l'utilisateur de résoudre le problème.

#### 5.3.4.2 Cas nominal

Tous les services sont fonctionnels et l'utilisateur commence à saisir des caractères dans la barre de recherche. Au bout de 0.5 seconde, si le contenu de la barre de recherche n'est pas modifié, la recherche est lancée. Lorsque la recherche se termine (succès ou non), on traite les résultats s'il y en a et on les affiche sous forme de liste. S'il n'y en a pas, un message d'erreur, non bloquant, est affiché.
Si le serveur ne répond pas (timeout) après 10 secondes, on affiche une erreur.
Pendant cette recherche, un spinner est affiché à l'écran afin de signaler l'activité à l'utilisateur.

#### 5.3.4.3 Fonctions importantes utilisées

- **tableView(*tableView: UITableView, numberOfRowsInSection section: Int*) -> Int** : calcule le nombre de lignes à afficher en fonction du nombre de résultats obtenus ; retourne ce nombre de lignes ; affiche un message si aucun résultat n'est trouvé
- **tableView(*tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath*) -> UITableViewCell** : rempli la cellule (la ligne) de la liste ; retourne une *UITableViewCell*, la cellule remplie avec les données

- **updateSearchResultsForSearchController(searchController: UISearchController)** : vérification si la barre de recherche n'est pas vide et attente d'une demi-seconde avant de lancer la recherche ; si la barre est modifiée, le timer est reset à nouveau et on attend à nouveau une demi-seconde avant de lancer la recherche
- **launchSearch()** : effectue la requête asynchrone auprès de Google Places, traite la réponse et créer les objets lieux
- **reloadAutoCompleteData()** : application d'un style sur le texte affiché dans les lignes de la liste
- **tableView(*tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath*)** : lance l'initialisation du chargement de la nouvelle vue lorsqu'une ligne est sélectionnée
- **prepareForSegue(*segue: UIStoryboardSegue, sender: AnyObject!*)** : chargement de la nouvelle vue en lui communiquant les paramètres de la ligne sélectionnée

### 5.3.5 Recherche de places à proximité d'un lieu

Cette fonctionnalité est similaire à la recherche par géolocalisation. C'est le contrôleur **SearchSelectedViewController** qui gère cette vue.
Les cas d'erreurs, le scénario nominal et les fonctions utilisées sont les mêmes que pour recherche par géolocalisation. Seuls quelques ajouts dans le scénario nominal et les fonctions sont effectués.

#### 5.3.5.1 Cas nominal

Après avoir sélectionné un lieu, la même vue que pour recherche par géolocalisation se charge. En même temps, la récupération des coordonnées du lieu est effectuée. Une fois effectué, un marqueur le représentant est ajouté sur la carte. L'utilisateur peut ainsi lancer la recherche (voir section 5.3.1.2), mais avec le lieu sélectionné comme source de recherche.

#### 5.3.5.2 Fonctions importantes utilisées

Les fonctions suivantes sont ajoutées, en plus des fonctions déjà décrites dans la section 5.3.1.2.

- **getCoordinate()** : effectue la requête asynchrone auprès de Google Places, traite la réponse et affiche le marqueur sur la carte
- **launchRecherche()** : remet les paramètres de recherche à zéro et initie la recherche

### 5.3.6 Multi langues

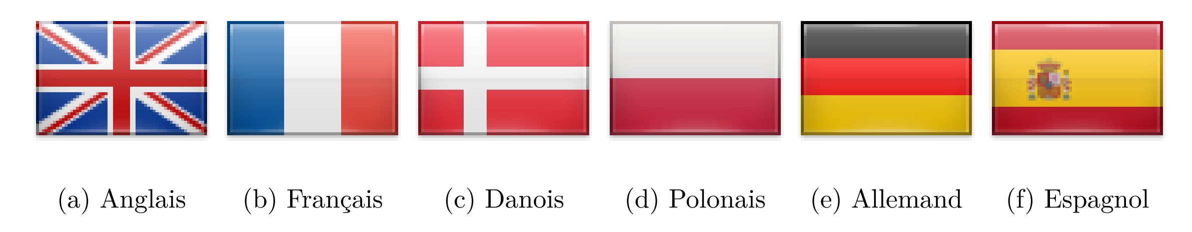
L'internationalisation (i18n) et la localisation (l10n) des applications sont essentielles quand on souhaite que celles-ci gèrent plusieurs langues de façon transparente.
Une fois le processus d'internationalisation (i18n) terminée, il suffit d'effectuer une itération de localisation (l10n) pour chaque langue que l'on souhaite ajouter.

L'application gère la localisation des textes et des dates (durée). L'utilisation d'icônes sans labels permet une compréhension universelle du message, évitant la multiplication d'images pour chaque langue.

#### 5.3.6.1 Localisation des textes

*Reddit* est un site regroupant de nombreuses communautés éparses, intéressées par des sujets aussi divers que les jeux vidéo, les chats mignons, la géopolitique mondiale ou l'athéisme. Friands

de contenus originaux, les utilisateurs du site ont la possibilité de voter pour les sujets de discussion qu'ils préfèrent, leur permettant d'atteindre la page d'accueil dans certains cas et donc d'accéder à une importante visibilité : 135 millions de personnes visitent le site par mois. Ainsi grâce à l'aide de plusieurs redditors, l'application a pu être rapidement traduite en plusieurs langues (voir figure 5.5).



(a) Anglais (b) Français (c) Danois (d) Polonais (e) Allemand (f) Espagnol

FIGURE 5.5 – Langues supportées par l'application

#### 5.3.6.2 Localisation des dates

Depuis iOS 8, une nouvelle classe est apparue dans le SDK d'iOS : *NSDateComponentsFormatter*.

*NSDateComponentsFormatter* formate une quantité de temps en une chaîne de caractères lisible par l'utilisateur. Cette classe a plusieurs options permettant de créer des chaînes plus ou moins longues en fonction du besoin. Pour l'application, une forme abrégée a été choisie. La classe prend également en compte la langue actuelle de l'utilisateur lors de la génération des chaînes de caractères.

## 5.4 Résultat

Les principales fonctionnalités du projet, définies dans le cahier des charges et explicitées plus amplement dans les spécifications détaillées, ont toutes été développées et sont fonctionnelles.

### 5.4.1 Contournements réalisés

Néanmoins, même si toutes les fonctionnalités sont opérationnelles, certaines spécifications ont dû être modifiées pendant la conception afin de se plier à certaines contraintes.

#### 5.4.1.1 Emplacement(s) proche(s) de ma position

Les spécifications prévoyaient un minimum de 5 emplacements, ce nombre est maintenant passé à 10. Un rayon de recherche intermédiaire, 5km, a également été rajouté.

#### 5.4.1.2 Récupération des informations complémentaires

Les spécifications prévoyaient de lancer la récupération des informations annexes, comme l'adresse approximative, après que la fin de la recherche, et ce pour chaque emplacement. Maintenant, la récupération de ces informations ne se fait que lorsqu'on souhaite afficher le détail d'un marqueur, permettant d'économiser du temps et de la bande passante.

#### 5.4.1.3 Affichage des informations complémentaires

Les spécifications prévoyaient l'affichage des informations annexes dans une nouvelle vue après avoir sélectionné le label d'un marqueur ; cette façon d'afficher les données étant assez

lourde pour l'utilisateur (temps de chargement de la vue à l'écran), il a été choisi d'afficher les informations dans label du marqueur en créant un label personnalisé.

#### 5.4.1.4 Source de récupération informations complémentaires

Les spécifications prévoyaient d'utiliser l'API *Google Distance Matrix* pour récupérer les informations annexes, mais cette API impose une limite très basse de requêtes (125 par jour), insuffisante pour l'application. Comme solution de substitution, il a été choisi d'utiliser *Google Maps SDK for iOS* pour récupérer l'adresse et *MapKit* pour la distance et la durée de trajet estimée, qui eux n'imposent aucune limite.

#### 5.4.1.5 Emplacement(s) proche(s) d'une adresse

Les spécifications prévoyaient le lancement de la recherche immédiatement après la saisie d'un caractère, mais uniquement après trois caractères saisis. Maintenant la recherche s'effectue dès le premier caractère, mais seulement lorsqu' aucun caractère n'est saisi pendant un délai de 0.5 seconde.

### 5.4.2 Nouvelles fonctionnalités

En plus des fonctionnalités prévues dans les spécifications, deux nouvelles fonctionnalités (présentes dans la liste de la section 4.5.2) ont été ajoutées.

#### 5.4.2.1 StreetView

*StreetView* est un service de Google permettant de visualiser un panorama à 360° d'un lieu situé sur une voie urbaine ou rurale.
*StreetView* étant inclus dans le SDK de Google Maps, son implémentation a été facilitée, n'ayant aucun composant externe à ajouter. *StreetView* requiert simplement les coordonnées géographiques, latitude et longitude. Une fois le panorama et la caméra configurés, on peut naviguer/zoomer dans *StreetView*.

#### 5.4.2.2 Informations complémentaires

À chaque emplacement fourni par OSM peuvent être associés des tags et ceux-ci peuvent se révéler utiles. Pour les utiliser, il suffit lors du traitement d'un emplacement de vérifier leur présence. La liste des tags utilisés est disponible dans la section A.6.

### 5.4.3 Outil non utilisé

Une des missions du stage était l'utilisation d'*AppCode*, une alternative à Xcode. Or *AppCode* est encore « peu » connu par rapport à Xcode, qui reste la référence dans les documentations et les tutoriels. De plus, Xcode est un IDE très complet et donc difficile à prendre en main. Partant de zéro dans le développement iOS, il a été choisi d'utiliser Xcode au lieu d'*AppCode*, pour un démarrage plus rapide.

## 5.5 Perspectives

À la fin des spécifications, une liste de fonctionnalités souhaitées a été rédigée, avant la phase de développement et donc sans la prise en compte des contraintes techniques. Avec le recul, voilà ce qu'on peut dire sur le degré de faisabilité de chacun des fonctionnalités :

- **emplacements sous forme de liste** : grâce à un bouton ou un segment de contrôle permettant de changer la vue en une *tableView*, il serait assez aisé d'implémenter cette fonction ; le même bouton permettrait de revenir sur la vue de la carte
- **favoris** : en utilisant *Core Data*, un framework d'Apple permettant de stocker de façon persistance des objets (à la manière d'une base de données), on pourrait facilement sauvegarder des emplacements. En revanche, il faudrait se méfier de l'obsolescence des données sauvegardées
- **filtrage grâce aux tags** : à l'aide de switchs et de sliders, on pourrait créer une vue de choix des paramètres, avec des paramètres par défaut
- **signaler/proposer/compléter un emplacement** : ces fonctions demandent une réflexion et un développement plus poussé ; l'API principale d'*OpenStreetMap* existe déjà, fonctionnant sous une architecture REST et avec OAuth, il faudrait implémenter les appels dans notre application, mais cela soulève de nombreux problèmes de synchronisation et des conflits (multiédition). Il faudrait également que les utilisateurs possèdent un compte utilisateur *OpenStreetMap* afin de pouvoir contribuer

Chapitre 6

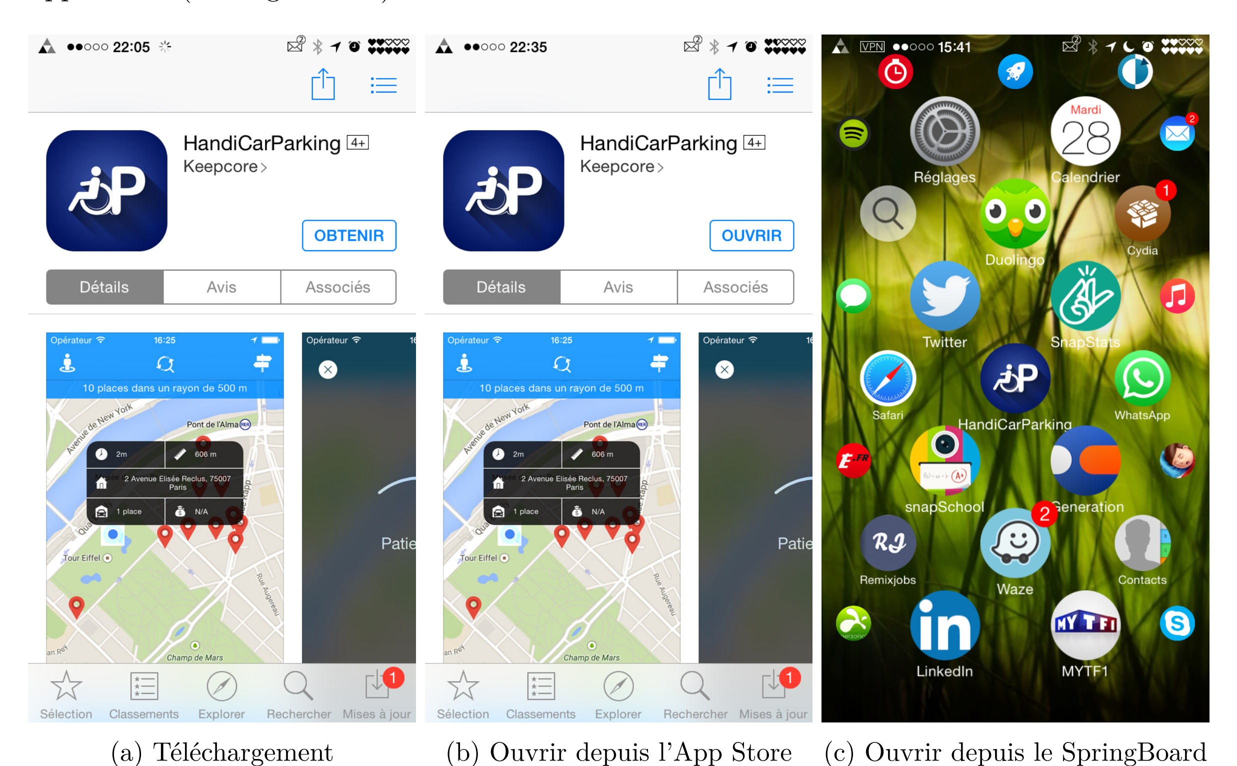
# Manuel d'utilisation

Le présent manuel d'utilisation n'explique que les fonctionnalités qui ont été développées.

## 6.1 Téléchargement & installation

Pour télécharger l'application, il suffit de rechercher `HandiCarParking` sur l'App Store ou de visiter directement ce lien depuis un appareil iOS : `https://itunes.apple.com/fr/app/id986777305`.

On arrive sur la page de présentation de l'application (voir figure 6.1a). Il suffit ensuite de simplement taper le bouton « *Obtenir* » pour lancer le téléchargement de l'application. Il est possible que vous deviez entrer les identifiants de votre compte Apple pour pouvoir télécharger l'application. Dès que le téléchargement est terminé, le bouton se modifie en « *Ouvrir* » (voir figure 6.1b). L'application peut-être directement lancée via ce bouton ou via l'icône de l'application (voir figure 6.1c).

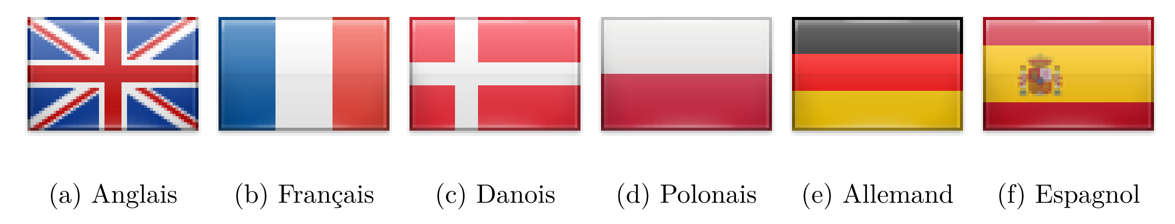



(a) Téléchargement (b) Ouvrir depuis l'App Store (c) Ouvrir depuis le SpringBoard

Figure 6.1

## 6.2 Présentation générale

L'application est disponible en 6 langues.



(a) Anglais (b) Français (c) Danois (d) Polonais (e) Allemand (f) Espagnol

FIGURE 6.2 – Langues supportées par l'application

L'application est composée de trois parties : la barre de navigation, le contenu de la vue et la barre d'onglets (voir figure 6.3).

### 6.2.1 Barre d'onglets

La barre d'onglets est l'équivalent de la barre de menu. Elle permet à l'utilisateur de naviguer rapidement entre les différentes vues de l'application. Elle se situe en bas de la fenêtre et est composée de trois onglets :

- **Géolocalisation**
- **Recherche**
- **À propos**

### 6.2.2 Barre de navigation

La barre de navigation contient les contrôles pour naviguer à travers les vues, effectuer des actions et gérer le contenu des vues. Elle se situe en haut de la fenêtre et son contenu changera en fonction de la vue.

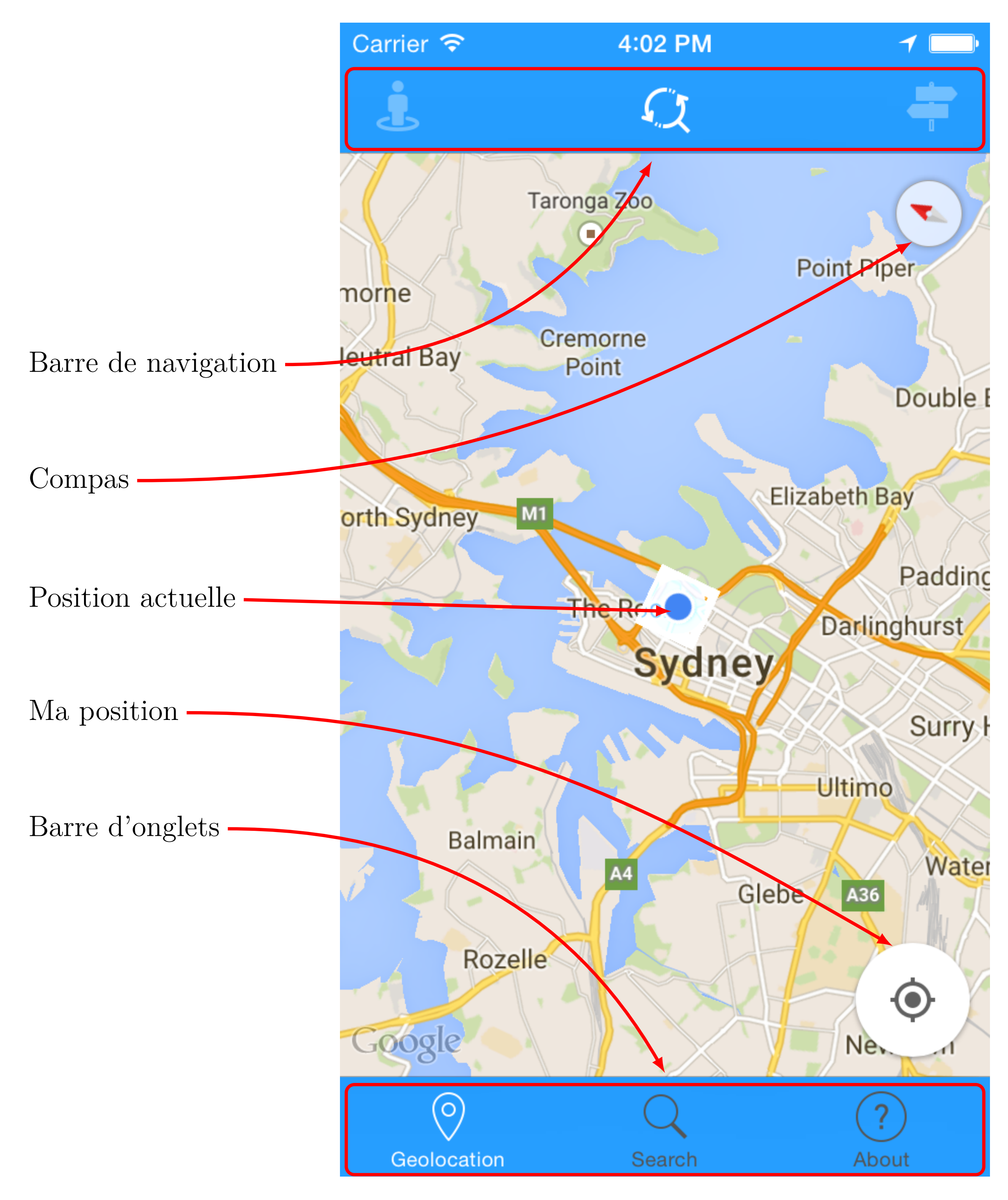
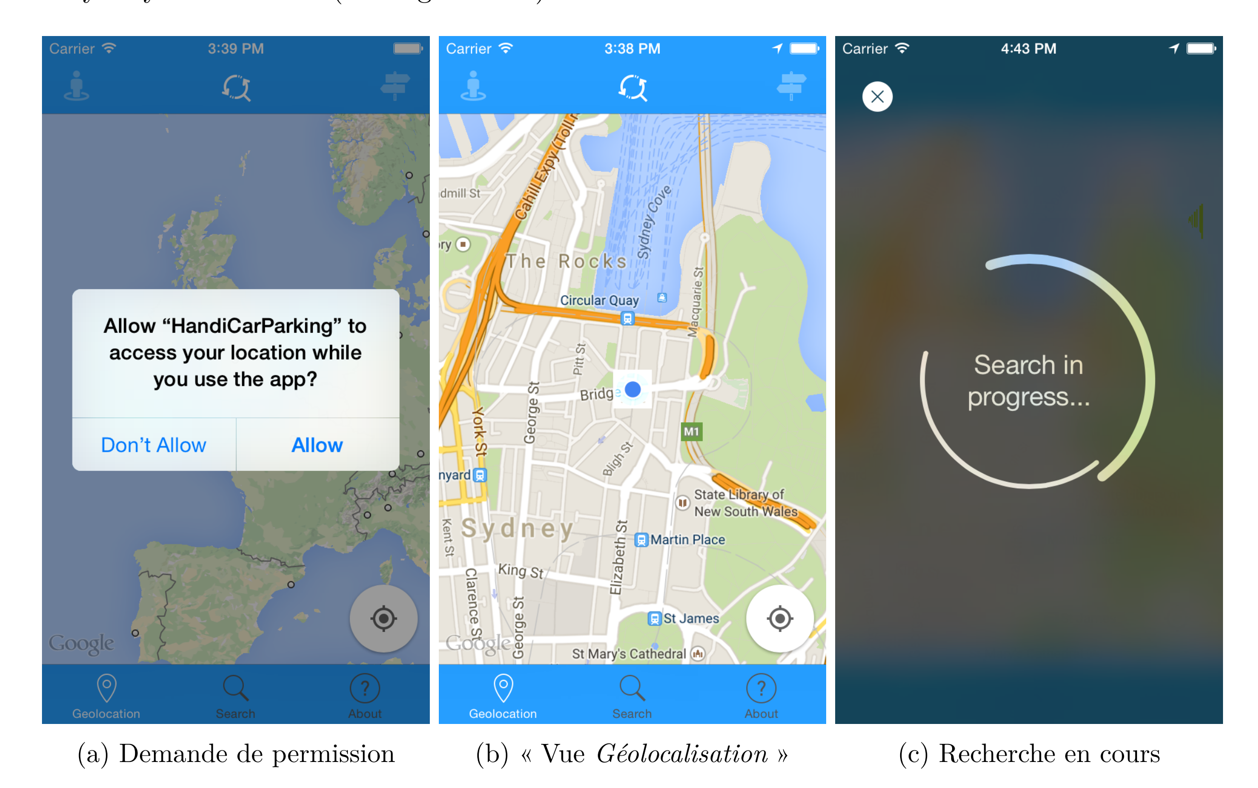



FIGURE 6.3 – Les différentes parties de l'application

## 6.3 Géolocalisation

Après avoir lancé l'application, l'utilisateur arrive directement sur la vue « *Géolocalisation* ». Si c'est son premier lancement, une alerte apparaît, lui demandant de partager ses données de géolocalisation (voir figure 6.4a).

Après avoir accepté, la carte va se charger de se centrer sur la position actuelle de l'utilisateur, ici Sydney en Australie (voir figure 6.4b).




(a) Demande de permission (b) « Vue *Géolocalisation* » (c) Recherche en cours

FIGURE 6.4

Depuis cette vue, trois boutons sont accessibles, mais un seul est activé : celui en forme de loupe, utilisé pour lancer la recherche.
En touchant ce bouton, la recherche se lance (voir figure 6.4c). Il est possible de l'arrêter à tout moment en touchant la croix en haut à gauche de la vue. Quand la recherche se termine, la carte s'affiche, remplie avec des marqueurs représentants les emplacements trouvés ainsi qu'un bandeau récapitulant les résultats de la recherche (voir figure 6.6a).

Les *gestures* habituels sont disponibles : zoom, dézoome, rotation et déplacement sur la carte. En touchant un marqueur, la carte se centre sur celui-ci, la fenêtre d'informations s'affiche et le chargement des informations se lance. Lorsque les informations sont récupérées, la fenêtre d'informations du marqueur se recharge et les informations sont affichées (voir figure 6.5).

En touchant un marqueur, on active également les deux autres boutons de la barre de navigation :

- le bouton **gauche** permet de lancer *StreetView* et ainsi avoir une vue réelle de l'emplacement trouvé. On peut se déplacer dans *StreetView*, zoomer, dézoomer très facilement (voir figure 6.6b). Pour fermer *StreetView*, il suffit de toucher la croix en haut à gauche.
- le bouton **droit** permet d'afficher une liste des applications capables de prendre en charge l'itinéraire jusqu'à cette place (voir figure 6.6c). En tapant une des applications de la liste, celle-ci se lance en mode navigation.

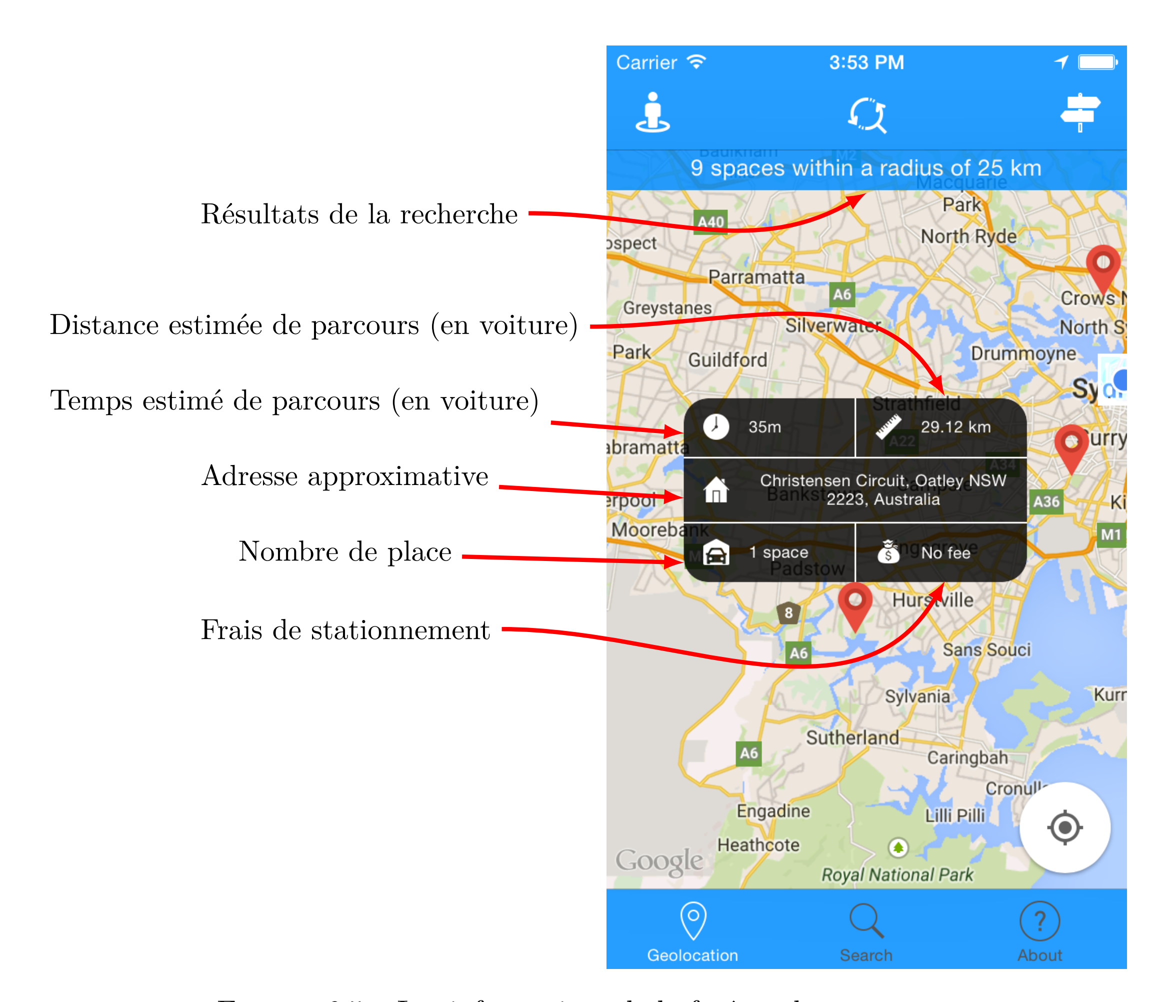
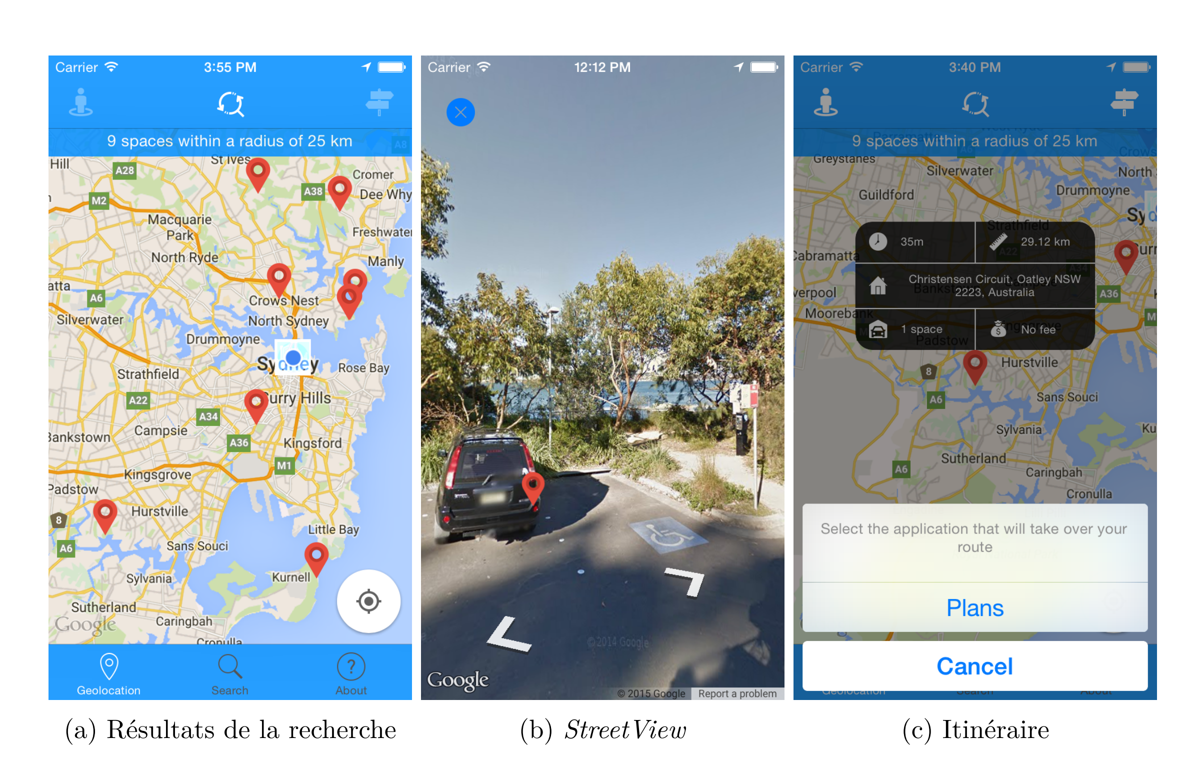



FIGURE 6.5 – Les informations de la fenêtre du marqueur

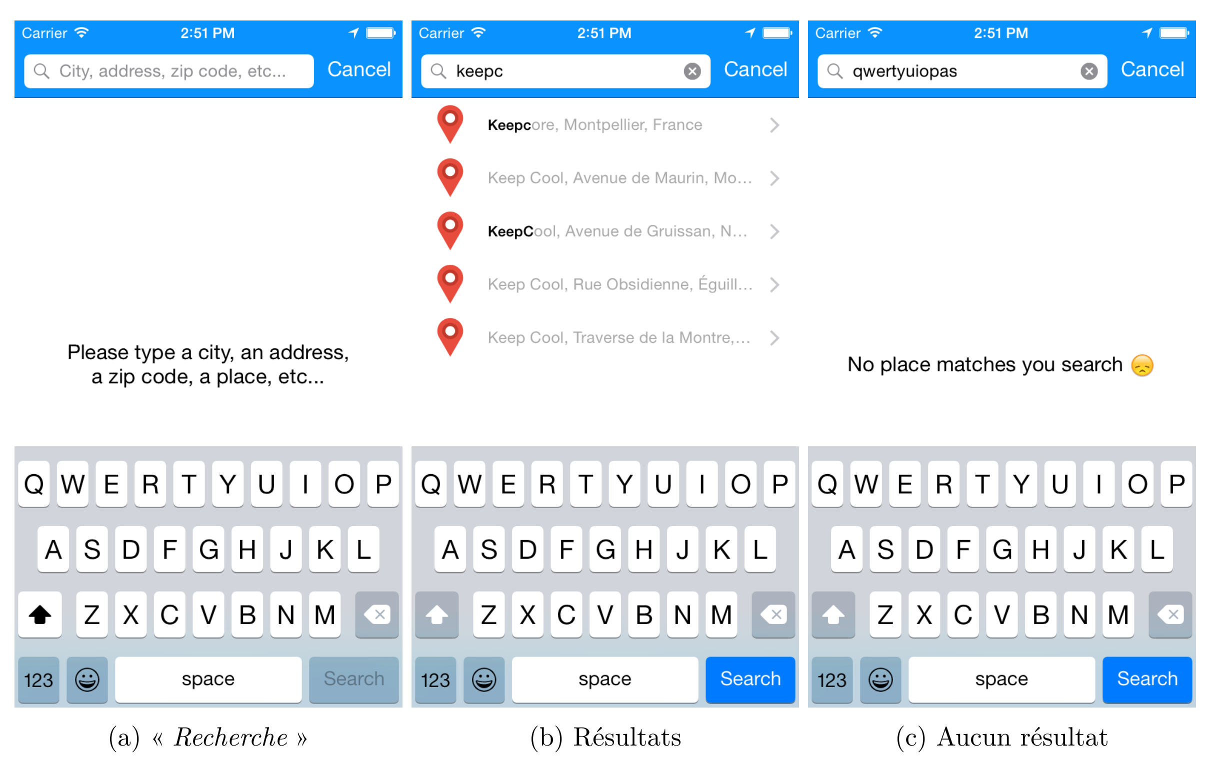



(a) Résultats de la recherche (b) *StreetView* (c) Itinéraire

FIGURE 6.6 – Actions complémentaires

## 6.4 Recherche

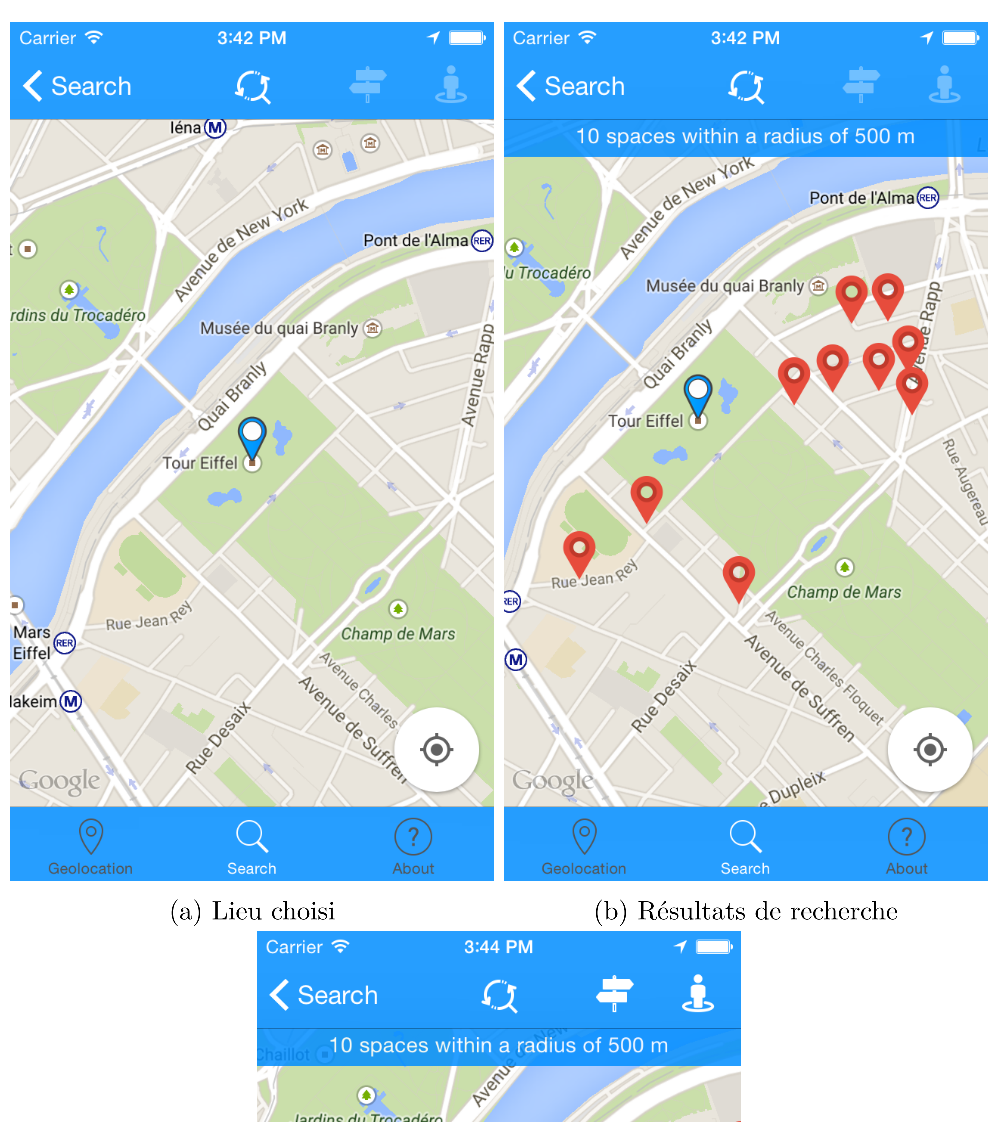
Pour arriver sur la vue de recherche, il suffit de taper sur « *Recherche* » dans la barre d'onglets. En arrivant sur la vue, l'utilisateur est directement mis en focus dans la barre de recherche et voit le clavier apparaître (voir figure 6.7a). Il n'a qu'à saisir des caractères pour que la recherche se lance automatiquement et obtenir les premiers résultats (voir figure 6.7b). Si aucun lieu ne correspond à sa recherche, un message est affiché (voir figure 6.7c).




(a) « *Recherche* » (b) Résultats (c) Aucun résultat

Figure 6.7

Pour choisir un lieu, il lui suffit de le toucher. Cette action va charger une nouvelle vue presque identique à la vue « *Géolocalisation* ». La carte de cette nouvelle vue va se charger se centrer sur la position correspondante au lieu choisi par l'utilisateur, ici la Tour Eiffel à Paris. L'utilisateur peut revenir s'il le souhaite à la vue de recherche grâce au bouton de retour à gauche de la barre de navigation ou avec un *swipe gesture.*
Pour tout le reste, les mêmes actions sont possibles : lancer la recherche d'emplacements, sélectionner un marqueur, *StreetView*, etc. (voir figure 6.8).




(a) Lieu choisi

(b) Résultats de recherche

(c) Marqueur sélectionné

FIGURE 6.8 – Recherche d'emplacements autour d'un lieu

Chapitre 7

# Rapport d'activité

Dans cette partie, je vais vous présenter dans un premier temps les différents logiciels, méthodes et outils que j'ai utilisés pour travailler sur ce projet. Dans un second temps, je parlerai de la planification et de la répartition du travail que j'ai effectué, en concertation avec mon tuteur, pour la gestion du projet.

## 7.1 Méthode de développement

Le projet *HandiCarParking* n'a pas été développé selon une méthode de développement formelle. La gestion du projet s'est faite à l'aide de divers outils de communication et de partage, mais sans outil réel de gestion comme *Trello*, *JIRA* ou gestionnaire de tickets. Des revues quasi journalières ont été effectuées, afin de faire le point sur l'avancement, les difficultés et les solutions possibles, que ce soit en terme de fonctionnalités ou d'ergonomie.

Ce choix a été fait plus ou moins implicitement par souci de simplicité : étant seul à travailler sur le projet, sans contraintes externes particulières mises à part la durée du stage, ayant tout à faire (apprentissage, analyse, conception, déploiement et documentation) et partant de zéro, aucune gestion globale n'était envisageable. Tout au long du projet, j'ai recentré les priorités afin de m'assurer de pouvoir avancer, avoir un résultat satisfaisant et ne pas rester bloqué.

## 7.2 Outils utilisés

### 7.2.1 Logiciel de versionning

#### 7.2.1.1 Subversion

Un logiciel de gestion de versions est un outil incontournable pour tout développeur. Il en existe beaucoup et KeepCore a choisi d'utiliser *Subversion* ou *SVN*.

SVN se base sur le mode client-serveur. C'est un système centralisé à la différence de Git, un autre logiciel de versionning, qui lui est décentralisé. Le dépôt (*repository*) constitue le cœur de ce système, en tant que lieu de stockage central des données. Les informations y sont organisées sous la forme d'une arborescence de fichiers, c'est-à-dire une hiérarchie classique de fichiers et de répertoires. Un certain nombre de clients se connectent au dépôt, et parcourent ou modifient ces fichiers.

Le fonctionnement de SVN est simple : chacun travaille sur les fichiers en local (sur son ordinateur), puis lorsqu'il pense avoir terminé la partie qu'il voulait développer, il commente et envoie les modifications effectuées (*commit*) sur le serveur (*repository*). Il est possible de revenir

en arrière s'il y a une erreur, de voir l'historique des modifications et les conflits sont gérés de façon presque automatique par SVN.

#### 7.2.1.2 Subversion dans le projet

Étant le seul à travailler sur le projet, je n'ai pas eu à me soucier des conflits et du verrouillage de fichiers. J'effectuais des commits régulièrement (maximum toutes les 2 h) afin de m'assurer d'avoir une forte cohérence tout au long du développement. Grâce à ce suivi régulier, découpant le développement en petites parties, je n'ai pas eu à effectuer de *rollback* (retour en arrière).

Le projet se compose de plus de **180 commits** pour un mois et demi de développement. Les figures 7.1 et 7.2 représentent des statistiques intéressantes, créées à partir des métadonnées des commits, sur la répartition et les habitudes de travail.

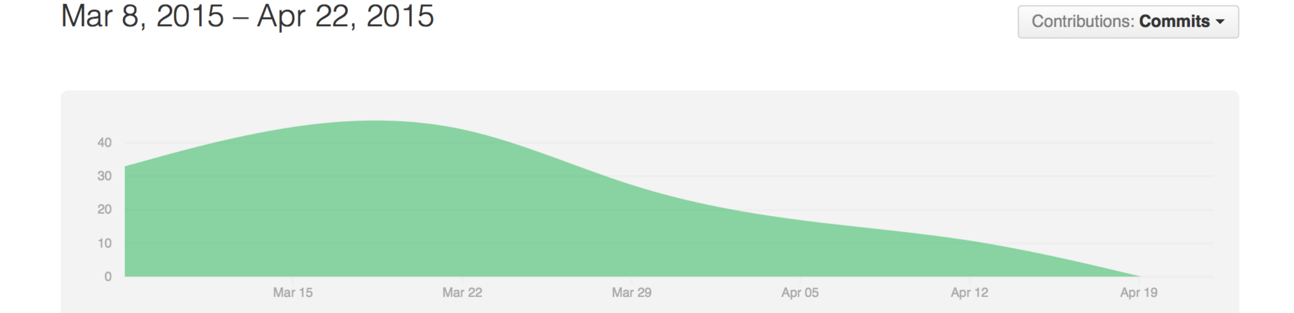



FIGURE 7.1 – Graphique du nombre de commits en fonction du temps

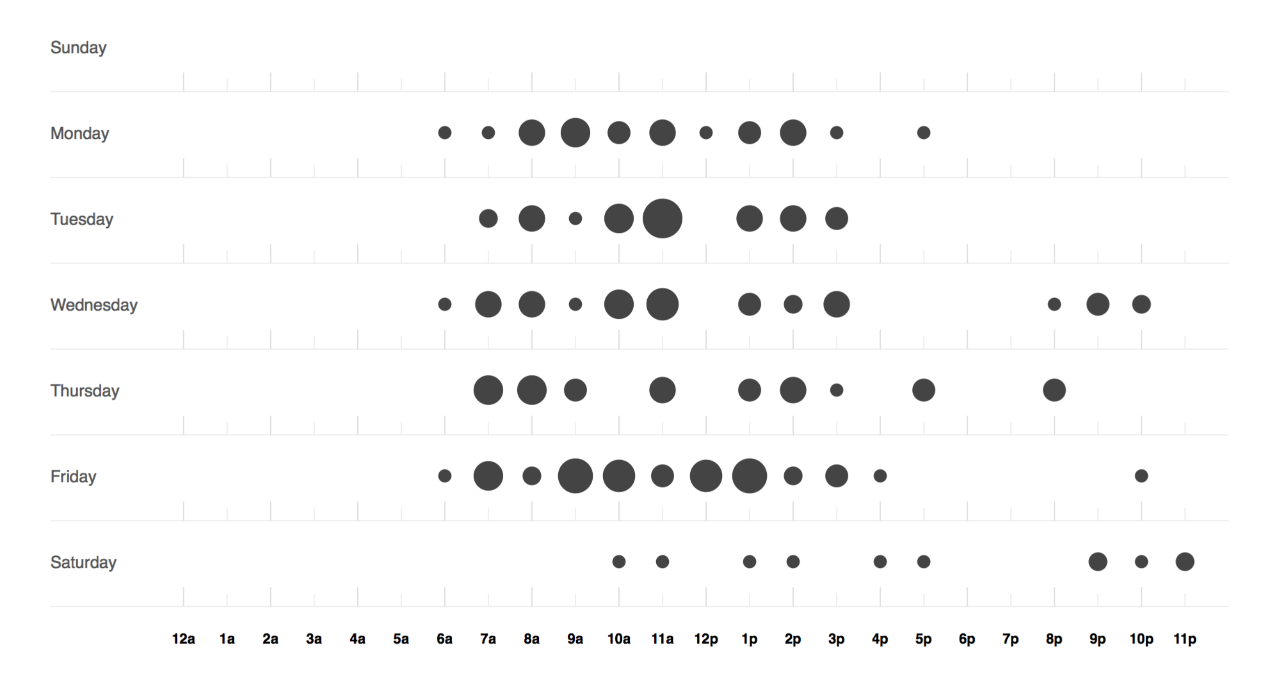



FIGURE 7.2 – Graphique de la fréquence des commits basé sur l'heure de la journée et le jour de la semaine

### 7.2.2 Xcode

*Xcode* est un environnement de développement pour OS X et iOS. Il permet notamment de développer en C++, Objective-C et Swift. Il est disponible gratuitement sur le Mac App Store et est fourni avec toute une suite d'outils.
*Xcode* regroupe dans son espace de travail à la fois un éditeur de code et un éditeur de fichiers d'interface, l'*Interface Builder*. Il possède également quantité d'options et de paramètres : débug, aide, documentation, versionning, etc. (voir figure 7.3)
C'est aussi avec Xcode qu'on effectue l'envoi de l'application vers la plateforme de validation avant publication sur l'App Store.

Il a été assez difficile à prendre en main les premiers jours, surtout la découverte de Swift à gérer en même temps. Mais dès que j'ai commencé à prendre mes repères, mon *workflow* s'est considérablement amélioré.

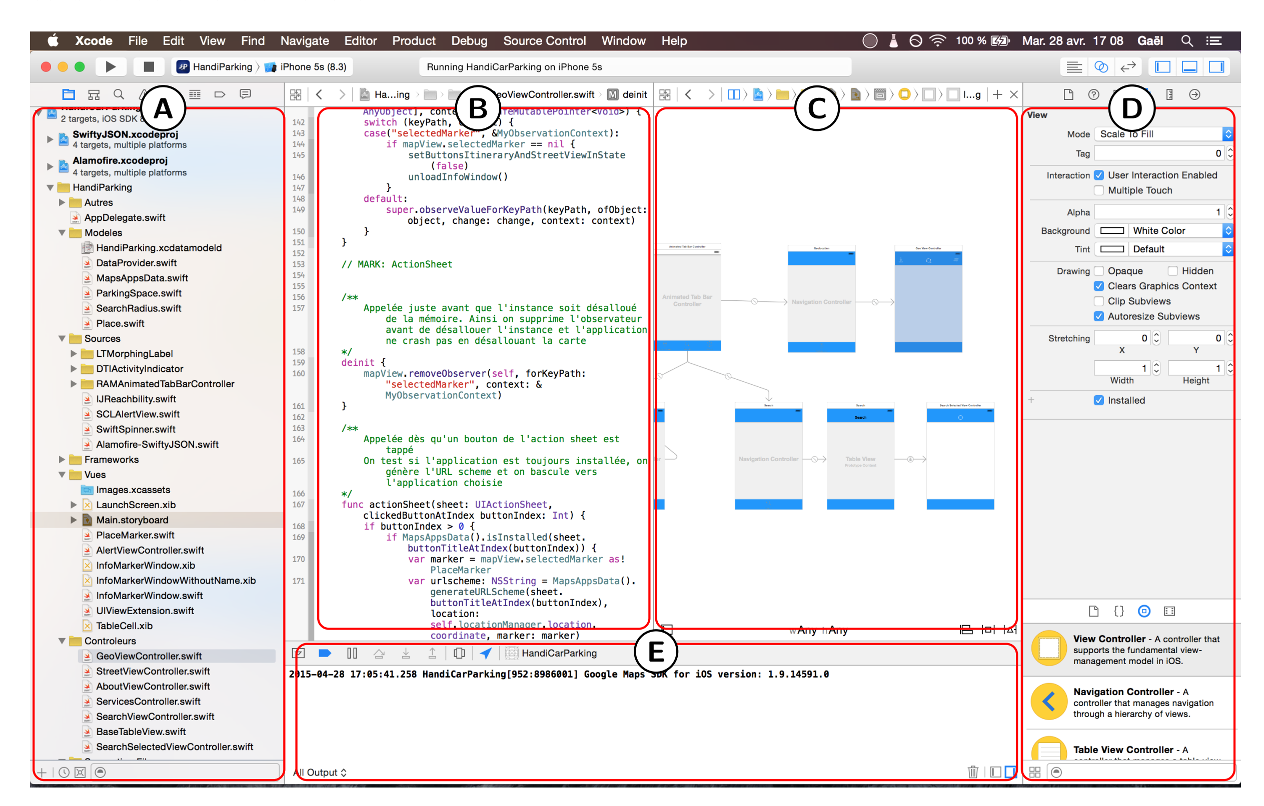



FIGURE 7.3 – **Les différentes parties d'Xcode**
Arborescence (A), Éditeur de code (B) Interface Builder (C)
Outils de l'Interface Builder (D) Console de debug (E)

### 7.2.3 Google Drive

*Google Drive* est l'outil utilisé par KeepCore pour rédiger ses documents (administratifs, techniques, etc.) et pouvoir les partager facilement. Ainsi tous les documents que j'ai rédigés pour les besoins du stage sont dans un dossier *Google Drive*, accessible à tous les salariés de KeepCore.
Cet outil a été un atout, car il a permis à Jérémy de suivre mon avancement lors de la rédaction des documents et d'effectuer un feedback continu en annotant et commentant les documents. Cela nous a évité une circulation trop importante d'emails.

### 7.2.4 Visual Paradigm

*Visual Paradigm* est un logiciel de conception UML très complet qui a servi à élaborer les diagrammes UML disponibles dans les spécifications (voir chapitre 4). J'ai utilisé la version « *Community* », gratuite pour un usage non commercial. Il est facile à prendre en main, intuitif et son site possède une excellente documentation.

## 7.3 Planification & répartition

Dès le début du projet, Jérémy m'a demandé d'élaborer un diagramme de Gantt, pour évaluer grossièrement les grandes lignes directrices du projet et ainsi laisser un peu de temps pour les imprévus (voir figure 7.4). Dans son ensemble, le planning a été respecté.

### 7.3.1 Semaines 1 et 2

Durant ces deux premières semaines, j'ai tout d'abord dû apprendre un nouveau langage, Swift. Cela a été ma principale tâche durant les deux premières semaines : je n'ai pas vraiment été très productif, lisant beaucoup de documentation et tutoriaux. Une synthèse du langage Swift a été rédigée pendant cet apprentissage. Parallèlement à ça, j'ai commencé à effectuer des recherches sur *OpenStreetMap* afin de me familiariser avec le projet, lisant là aussi beaucoup le wiki. Une synthèse des résultats de l'étude est disponible en annexe. J'ai également effectué un état de l'art, que j'ai plus tard intégré aux spécifications.

### 7.3.2 Semaines 3 et 4

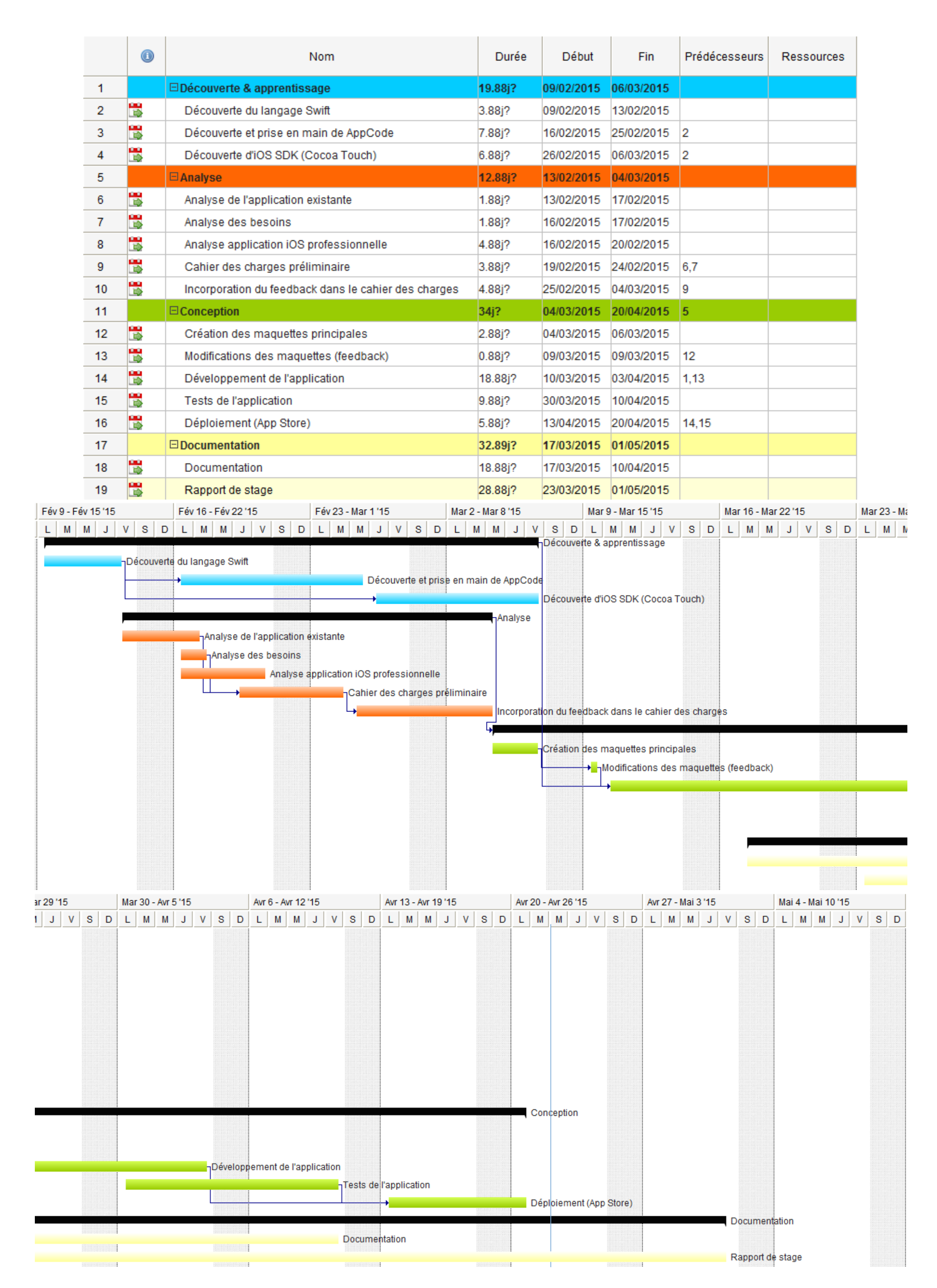
Les deux semaines suivantes, j'ai commencé à me familiariser avec Xcode et par la même occasion le SDK d'iOS. En même temps que ça, j'ai rédigé le cahier des charges du projet, ayant pu synthétiser les bonnes idées vues dans l'état de l'art. Juste après ça, je me suis attaqué à la rédaction des spécifications détaillées. Grâce à l'étude sur *OpenStreetMap*, l'analyse du besoin et ma connaissance du SDK d'iOS, j'ai pu choisir les fonctionnalités que je pensais pouvoir réaliser dans le temps que je m'étais accordé pour le développement, un peu plus d'un mois. Une étude du design d'une application iOS a également été commencée et mise à jour tout au long du développement.

### 7.3.3 Semaines 5 à 9

Ces cinq semaines ont été l'objet du développement de l'application. Le démarrage a été un peu lent, le temps de mettre en place les outils, l'architecture de l'application et se familiariser avec SVN. Je me suis heurté à quelques problèmes que je n'avais pas détectés dans la phase d'analyse et après concertations avec Jérémy, des solutions équivalentes ont été trouvées.
La dernière semaine de développement a été l'objet de correction de bugs en tout genre, d'un lissage de l'interface afin d'obtenir un résultat homogène lors de la navigation, et de l'internationalisation de l'application.
Au milieu de ce développement, le rapport a été mis en place. J'ai utilisé *ShareLaTeX*, un site de rédaction en ligne de document au format LaTeX et qui offre la possibilité d'éditer à plusieurs de façon instantanée. Les premières parties déjà rédigées ont été progressivement intégrées au rapport.

### 7.3.4 Semaines 10 à 12

Ces dernières semaines ont été consacrées principalement à la rédaction du rapport. L'étude sur le design d'une application iOS a été terminée et intégrée.
La première semaine a été en partie utilisée pour le déploiement de l'application sur l'App Store.



| | | Nom | Durée | Début | Fin | Prédécesseurs | Ressources |
|---|---|---|---|---|---|---|---|
| 1 | | ⊟Découverte & apprentissage | 19.88j? | 09/02/2015 | 06/03/2015 | | |
| 2 | | Découverte du langage Swift | 3.88j? | 09/02/2015 | 13/02/2015 | | |
| 3 | | Découverte et prise en main de AppCode | 7.88j? | 16/02/2015 | 25/02/2015 | 2 | |
| 4 | | Découverte d'iOS SDK (Cocoa Touch) | 6.88j? | 26/02/2015 | 06/03/2015 | 2 | |
| 5 | | ⊟Analyse | 12.88j? | 13/02/2015 | 04/03/2015 | | |
| 6 | | Analyse de l'application existante | 1.88j? | 13/02/2015 | 17/02/2015 | | |
| 7 | | Analyse des besoins | 1.88j? | 16/02/2015 | 17/02/2015 | | |
| 8 | | Analyse application iOS professionnelle | 4.88j? | 16/02/2015 | 20/02/2015 | | |
| 9 | | Cahier des charges préliminaire | 3.88j? | 19/02/2015 | 24/02/2015 | 6,7 | |
| 10 | | Incorporation du feedback dans le cahier des charges | 4.88j? | 25/02/2015 | 04/03/2015 | 9 | |
| 11 | | ⊟Conception | 34j? | 04/03/2015 | 20/04/2015 | 5 | |
| 12 | | Création des maquettes principales | 2.88j? | 04/03/2015 | 06/03/2015 | | |
| 13 | | Modifications des maquettes (feedback) | 0.88j? | 09/03/2015 | 09/03/2015 | 12 | |
| 14 | | Développement de l'application | 18.88j? | 10/03/2015 | 03/04/2015 | 1,13 | |
| 15 | | Tests de l'application | 9.88j? | 30/03/2015 | 10/04/2015 | | |
| 16 | | Déploiement (App Store) | 5.88j? | 13/04/2015 | 20/04/2015 | 14,15 | |
| 17 | | ⊟Documentation | 32.89j? | 17/03/2015 | 01/05/2015 | | |
| 18 | | Documentation | 18.88j? | 17/03/2015 | 10/04/2015 | | |
| 19 | | Rapport de stage | 28.88j? | 23/03/2015 | 01/05/2015 | | |



FIGURE 7.4 – Diagramme de Gantt

J'ai du me familiariser avec le processus de déploiement d'applications d'Apple et préparer toutes l'application au processus de validation. Une fois fait, j'ai pu remplir la page de présentation de l'application qui sera visibe sur l'App Store. En même temps, des fiches d'utilisation d'outils permettant d'aider au test et au débug lors du développement d'application iOS ont été rédigées pour un éventuel développement futur.

CHAPITRE 8

# Conclusion

L'objectif de ce stage était de rendre accessible sous une forme simple, **la localisation des places de parkings réservées aux personnes handicapées**. Pour cela, une application mobile a été développée.

En prenant comme paramètre la position de l'utilisateur ou la position d'un lieu de son choix, l'application est capable de récupérer la localisation des places autour de cette position et de les afficher sur une carte.

**OpenStreetMap** a été choisi par Jérémy comme référentiel de données pour notre application en raison de son aspect open source, qui s'inscrit dans une démarche de transparence et de participation des citoyens, tendance se manifestant particulièrement ses dernières années.

Lors de l'écriture de ce rapport, *OpenStreetMap* référençait **21 335** places de parkings réservées aux personnes handicapées dans le monde. Néanmoins, ce référencement est très incomplet et la répartition des places est très hétérogène (voir figure 8.1).

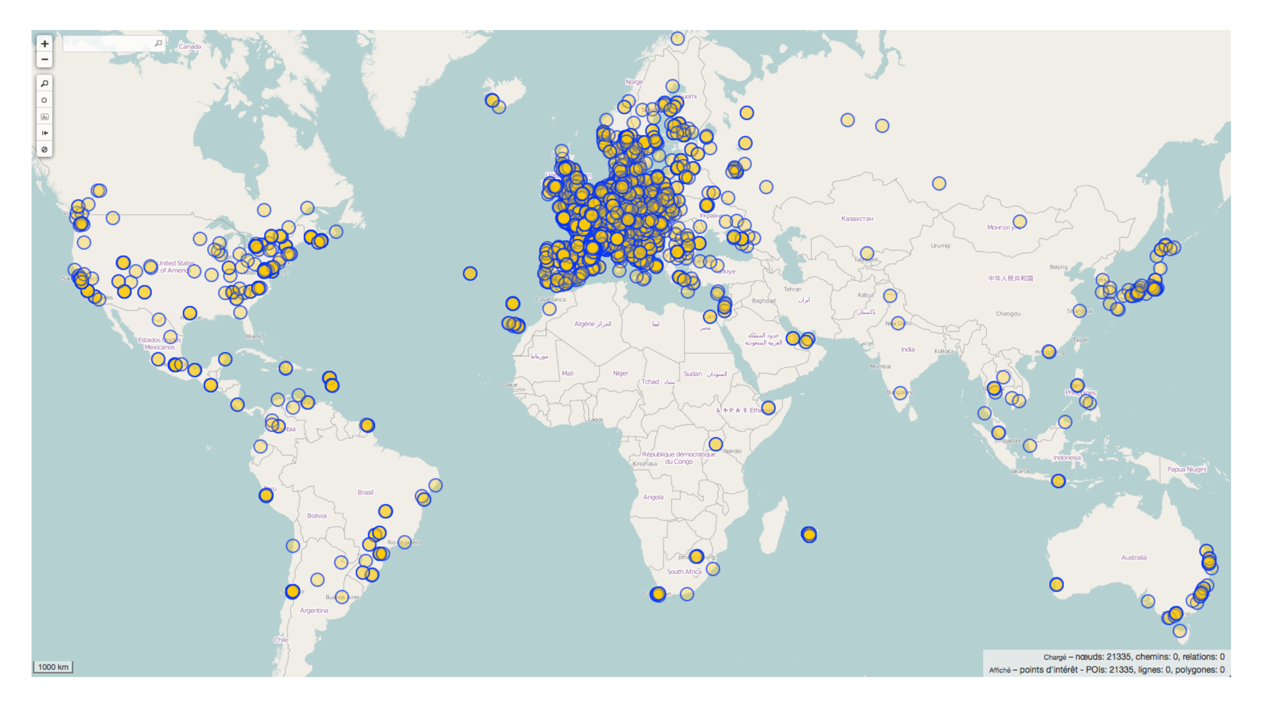



FIGURE 8.1 – Répartition des places de parkings pour personnes handicapées référencées dans *OpenStreetMap*

La plus grande majorité des places se trouvent en Europe et la France en compte à elle seule plus de **6 520**, soit près d'**un tiers** des places référencées.

La plate-forme cible de l'application, **iOS**, a également été choisie par Jérémy. Cette contrainte a demandé l'apprentissage d'un nouveau langage, **Swift**, de l'**iOS SDK**, ainsi que de tous les outils utiles gravitant autour de cette plate-forme.

Ce stage a été, plus que tout, une expérience très enrichissante, autant sur le plan personnel que professionnel.

Il m'a tout d'abord permis d'explorer la **problématique de l'open data**. La philosophie portée par ce mouvement est à l'origine profondément citoyenne. L'appropriation collective et associative de l'open data doit servir à dynamiser la vie publique et contribuer à réinventer les services publics : transparence, participation, coproduction et confiance sont au cœur de la démarche de l'open data. *OpenStreetMap* est un des meilleurs exemples de ce mouvement et est devenu au fil des années une référence dans le monde la cartographie [6], jusque là réservée à des spécialistes.

J'ai également découvert le **monde du développement sur mobile**. Cette branche du développement est en pleine expansion ces dernières années ce qui en explique son attrait. Les outils, modèles et règles de conceptions utilisées évoluent chaque jour. De nouvelles plates-formes apparaissent, comme la *Pebble Time* ou l'*Apple Watch*, offrant quantité de nouvelles possibilités. Tout au long du stage, j'ai eu l'occasion de découvrir de nouvelles méthodes de développement, de nouvelles façons d'organiser mon code ou de gérer un projet. J'ai pu me rendre compte que je n'avais fait qu'effleurer la surface des possibilités offertes par les outils que j'ai utilisés, car les applications mobiles offrent une très grande interaction avec les utilisateurs.

Un des problèmes majeurs auquel j'ai été confronté, au début du stage, a été l'absence d'une base sur laquelle m'appuyer. Certes, j'avais des connaissances, abstraites, qui s'appliquent à toutes les formes de développement, mais je n'avais aucune compétence spécifique liée au sujet du stage. Je suis parti de zéro, sans aucune connaissance sur l'open data ou le développement mobile, pour créer cette application qui est **maintenant disponible à tous, et ce gratuitement.**

Néanmoins, elle est encore très simple et mérite de nombreuses améliorations et ajouts de fonctionnalités. Le **filtrage de la recherche** via des critères, l'**ajout/modification/suppression de places** directement depuis l'application ou encore l'attribution d'un **statut occupé/libre** de la place sont des pistes envisageables.

Le **géorepérage** (ou *geofencing*), permettant de créer des notifications lorsqu'on pénètre ou quitte une zone géographique définie, est également une piste intéressante à explorer. Il pourrait servir à avertir l'utilisateur de la disponibilité d'une place proche, la marquer comme occupée/libre de façon automatique lorsque l'utilisateur se gare/s'en va.

Cette dernière fonctionnalité pourrait plus généralement s'appliquer à tous types de parkings. En utilisant une application de gestion de parc de stationnement avec géorepérage, le gérant et le client seraient notifiés en temps réel de la disponibilité de places, permettant d'éviter conflits et pertes de temps.




# Recherches et études sur OpenStreetMap

Le projet consiste à développer une application mobile ayant pour objectif de faciliter l'accès aux places de parkings réservées aux personnes en situation de handicap dans le monde entier.

Pour cela, on se propose d'utiliser le projet *OpenStreetMap*. OSM est un projet ayant pour but de créer une base de données géographique libre et mondiale et peut se comparer à un « *Wikipédia pour cartes* », permettant à ses utilisateurs de collaborer en corrigeant et mettant à jour les informations. Les données collectées sont très variées, de l'emplacement des pylônes électriques à la largeur des trottoirs. Dans notre cas, les emplacements de places de parking réservées au handicapé. Les données extraites d'OSM sont disponibles à l'utilisation sous licence ODbL [4].

## A.1 Problèmes

Le premier problème posé est le suivant : on souhaite récupérer la localisation (latitude et longitude) des places de parking handicapé se situant près de la position actuelle, ainsi que des informations complémentaires y étant associées (désignation, payante, etc.). On suppose ici que l'on a déjà récupéré la localisation actuelle (longitude et latitude) et défini la zone de recherche à couvrir.

L'autre fonctionnalité qui serait intéressante à implémenter serait la localisation (latitude et longitude) des places de parking handicapé se situant près d'un endroit nommé (le nom de la ville, le code postal ou une adresse), ainsi que des informations complémentaires y étant associées (désignation, payante, etc.).
Sauf qu'ici un autre problème se pose : la recherche par ville peut retourner plusieurs résultats et il est possible que ce soit le cas aussi pour différents pays.

La solution à implémenter est donc d'afficher la liste des résultats trouvés. On va utiliser *Google Place Autocomplete* qui renvoie **5 résultats maximum** (il est arrivé d'en obtenir 6 lors des essais). Il ne reste ensuite qu'à récupérer la latitude et la longitude du lieu choisi et on se retrouve avec les mêmes données que le premier problème.

Les deux problèmes convergent vers une même solution, ce qui sera plus simple à implémenter et implique moins de refactoring.

## A.2 Solutions

Pour récupérer ces informations, on va utiliser l'une des API d'OpenStreetMap, qui se nomme *Overpass* (ou **OSM3S**). Contrairement à l'API principale qui est optimisée pour l'édition, *OSM3S* est une API de **lecture seule**. On pourrait la comparer à une base de données : le

client envoie une requête à l'API et celle-ci lui renvoie les données correspondantes à sa requête. Elle est optimisée de ce fait pour répondre aux requêtes avec une latence la plus faible possible. En revanche, le service est optimisé pour répondre à des requêtes sur des petites surfaces, il est préférable d'éviter les requêtes sur de grandes surfaces (utiliser un autre service).

Il existe également une autre API en lecture seule, nommée *XAPI*. Pourquoi avoir choisi **Overpass** plutôt que *XAPI* ? Tout simplement, car *Overpass* est plus complet, il supporte des requêtes avec des syntaxes complexes. De plus, Overpass gère la plupart des requêtes XAPI, le choix a donc été fait d'utiliser Overpass.

Overpass supporte deux types de syntaxes pour ses requêtes : *XML* et *QL*. La syntaxe **QL** a été choisie, car elle est plus légère, supporte les expressions régulières et plus simples à gérer dans l'application iOS que la syntaxe XML.

En revanche, les informations trouvées sur la fraîcheur et la validité des données étant assez disparates et contradictoires, on peut seulement en tirer les conclusions suivantes : dans le meilleur des cas les requêtes sont effectuées sur des dumps ayant quelques minutes et dans le pire des cas, on peut envisager qu'elles sont effectuées sur des dumps ayant quelques jours, une semaine maximum, ce qui reste très raisonnable.

## A.3 OpenStreetMap

Les éléments fondamentaux d'*OpenStreetMap* sont organisés en trois types : les nœuds (**nodes**), les chemins (**ways**) et les relations (**relations**). Chaque élément peut être associé à des **tags**, ceux-ci aidant à la compréhension de l'élément.

### A.3.1 Node

Une node définie un point spécifique dans l'espace grâce à sa latitude et à sa longitude. Une node est composé au minimum d'un identifiant, d'une longitude et d'une latitude.

### A.3.2 Way

Une way est une liste ordonnée de nodes (entre 2 et 2000 nodes). Les ways sont utilisés pour représenter les lignes, par exemple les routes. On peut également représenter des zones, comme des forêts, avec un way. Dans ce cas, le premier et le dernier node sont les mêmes.

### A.3.3 Relation

Une relation est une structure de données qui référence une relation entre deux ou plusieurs éléments (node, way et/ou autre relation). Par exemple, une autoroute est une relation de ways.

### A.3.4 Tag

Tout élément peut avoir des tags. Les tags décrivent les éléments auxquels ils sont attachés. Un tag consiste simplement à une association *clé=valeur*, avec des valeurs différentes et des options.

**Pour notre projet, nous ne ferons donc que des requêtes sur des nodes, puisque nous ne souhaitons obtenir que des points spécifiques dans l'espace.**

## A.4 Serveurs publics

- `www.overpass.osm.rambler.ru` (8 cœurs — 64 Go RAM)
- `www.overpass-api.de` (4 cœurs — 32 Go RAM)
- `www.api.openstreetmap.fr`
- `www.oapi-fr.openstreetmap.fr` (uniquement des données françaises)

Au moment de cette étude, le premier serveur de la liste semble être le plus rapide à répondre aux requêtes. Le dernier en revanche ne sera pas utilisé, car il ne fonctionne pas pour des requêtes portant sur une zone qui n'est pas française. Un switch entre les serveurs est à prévoir, au cas où un tomberait en panne ou serait surchargé.

## A.5 Overpass QL

*Overpass QL* est le second langage de requête pour l'API *Overpass* et a été conçu comme alternative à *Overpass XML*. Il est basé sur le style de syntaxe du *langage C* : la requête est divisée en instructions et chaque instruction se termine par un point virgule. Il a une sémantique impérative : les instructions sont traitées les unes après les autres et changent l'état de l'exécution en fonction de leurs sémantiques. *Overpass QL* supporte les **expressions régulières**.
Le tableau A.1 détaille quelques unes des instructions utiles à notre problème.

| Utilité | Syntaxe | Exemple(s) |
|---|---|---|
| Choisir le format de sortie des données | [out:format] | [out:json] ; [out:xml] ; [out:custom] |
| Tag(s) | [cle=valeur] ; ['cle'='valeur'] ; ["cle"="valeur"] ; ["cle:option"="valeur"] | [amenity=parking] [wheelchair=yes] ; ["capacity:disabled" ~"yes\|[0-9]"] |
| Zone (carré) à fouiller | node(sud,ouest, nord,est) | node(43.609401,3.869619, 43.610001,3.869719) |
| Zone (rond) à fouiller | node(around:rayon(mètre), latitude,longitude) | node(around:50, 43.609416,3.869641) |
| Affichage (output) | out *option option option etc* | out meta 50 |

Table A.1 – Exemple de tags - *Overpass QL*

Il en existe beaucoup d'autres, le filtrage par *ways*, la récursion, etc. mais qui ne nous intéressent pas pour ce projet.

## A.6 Tags à utiliser et combiner

Le wiki *OpenStreetMap* recensant les tags sur le handicap : `www.wiki.openstreetmap.org/wiki/Disabilities`

- **amenity=parking** : identifier une aire de stationnement pour des voitures, des camions, des motos, etc.

- **capacity:disabled=yes/no/nombre** : spécifie si un emplacement, une aire, etc. pour les personnes handicapées est disponible
- **amenity=parking_space** : signaler un emplacement particulier au sein d'un lot de places de parking
- **parking_space=disabled** : signaler un emplacement handicapé au sein d'un lot de places de parking
- **wheelchair=yes/no/limited** : marquer des lieux appropriés à l'usage des fauteuils roulants et/ou la présence d'une place de stationnement pour personnes à mobilité réduite
- **fee=yes/no** : précise s'il est nécessaire de payer pour un service ou un accès

## A.7 Statistiques

De part son aspect collaboratif, OSM est en permanence enrichi par les contributions des utilisateurs (voir figure A.1).

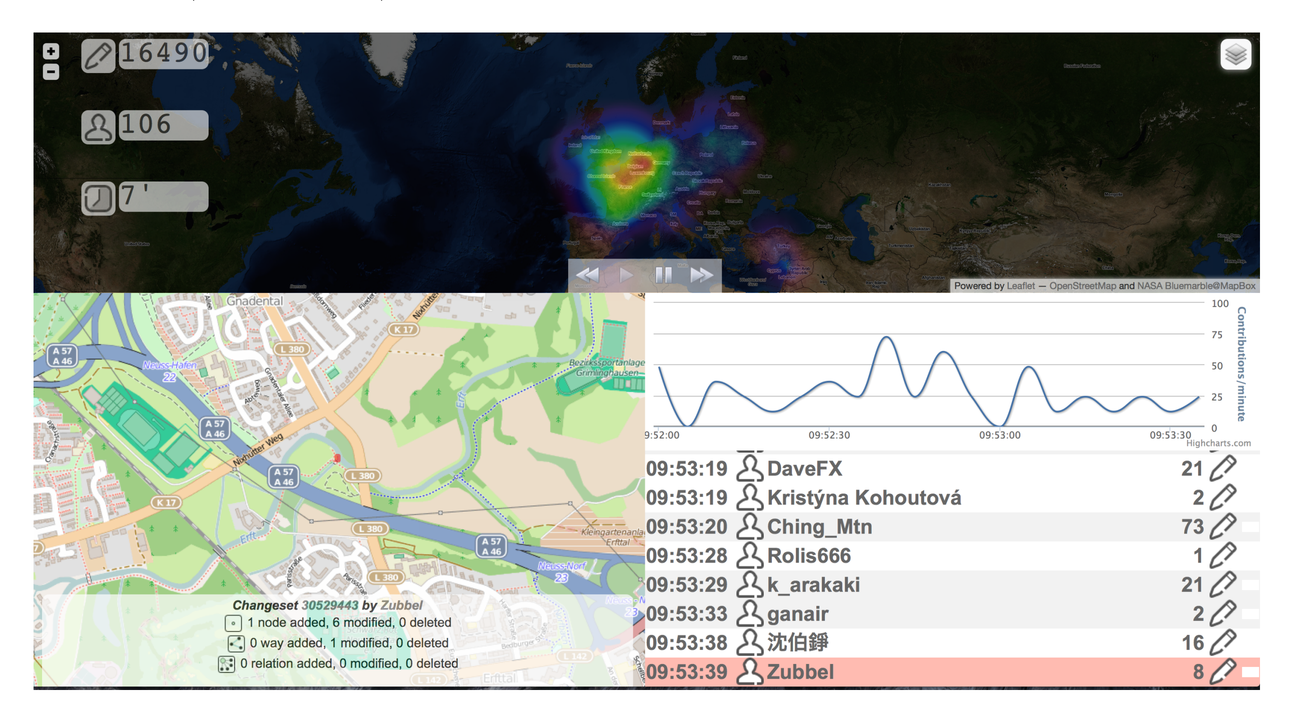



FIGURE A.1 – 16490 modifications effectuées par 106 utilisateurs dans les 7 dernières minutes sur OpenStreetMap




# Design iOS 8

## B.1 À propos

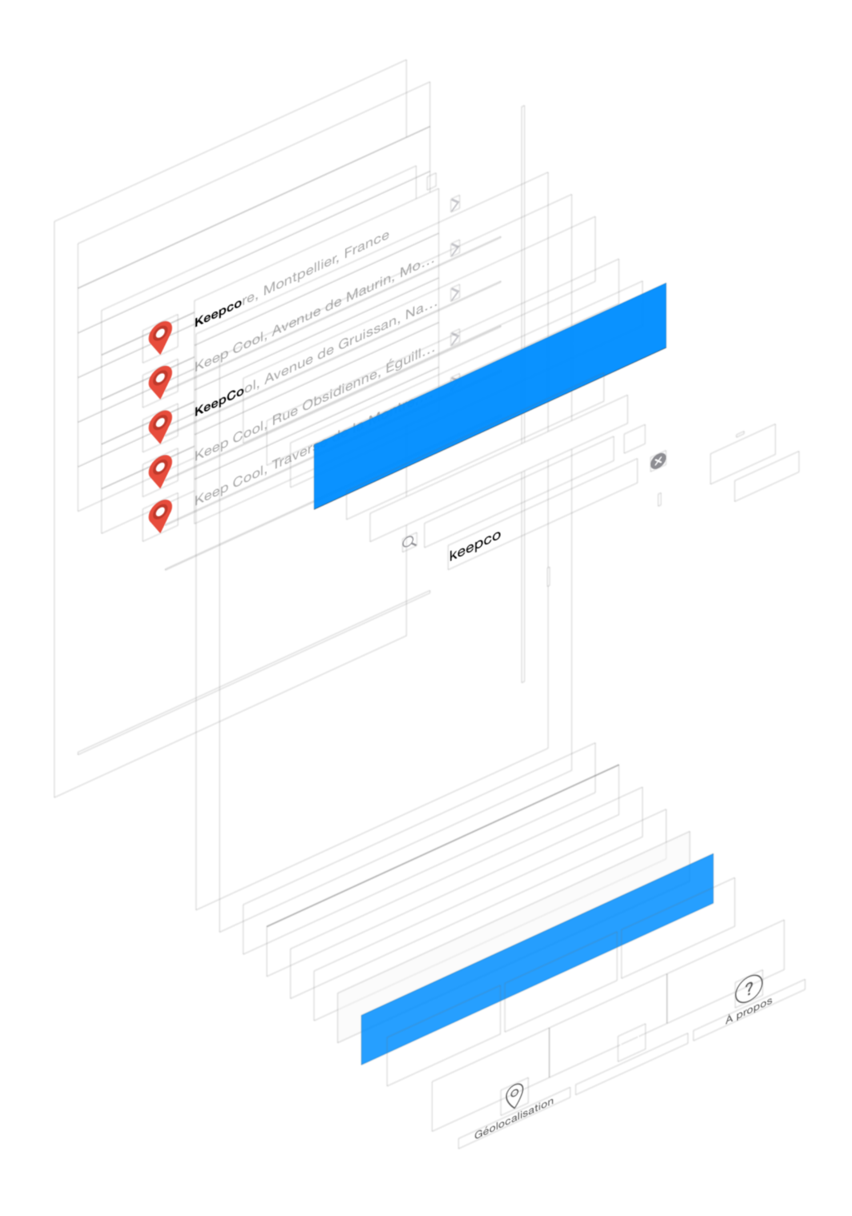



FIGURE B.1 – Hiérarchie des vues

Pour la sortie d'**iOS 7**, *Apple* a complètement changé ses idéaux concernant le design. Néan-

moins, si vous avez déjà conçu sur d'autres plateformes, le Web par exemple, les mêmes notions s'appliquent.
Les guidelines officielles d'*Apple* (HIG) concernant *iOS 8* est un document que tous les développeurs devraient lire et je le recommande, afin de se donner une vision d'ensemble des nouvelles règles de design d'*iOS 8*.
Ce document décrit les règles, les notions et les éléments principaux du design sous iOS 8, sans entrer dans les détails de conception. Le but de ce document n'est pas de fournir des solutions à des problèmes de design complexes et uniques, mais plus de donner une vue générale des éléments à notre disposition.

## B.2 Notions clés

*iOS 8* est dirigé par 3 notions clés : **déférence**, **clarté** et **profondeur**.

### B.2.1 Déférence

Le contenu doit être l'élément central, tout le reste est secondaire. Il faut éviter les éléments visuels distrayants qui peuvent gêner la vision du contenu. À chaque fois qu'un élément est ajouté, il faut se demander : est-il vraiment nécessaire ? Se concentrer sur l'harmonisation des couleurs, des dégradés et des typographies tout en évitant les effets 3D et les ombres.

### B.2.2 Clarté

Les éléments doivent être évidents, intuitifs. Les boutons doivent s'expliquer d'eux-mêmes et les typographies être grandes et lisibles de loin.

### B.2.3 Profondeur

Sûrement la notion la plus difficile à comprendre. C'est un concept abstrait, mais puissant. C'est l'idée que tout doit être contextuel. Dans la vie, on a une impression de progression et de distance. Le même concept doit s'appliquer à l'interface utilisateur.

## B.3 Résolution et affichage

| **Appareil** | **Retina** | **Portrait** | **Paysage** | **Résolution** | **Taille** |
|---|---|---|---|---|---|
| iPhone 6+ | Oui | 1080x1920 | 1080x1920 | @3x | 5.5" |
| iPhone 6 | Oui | 750x1334 | 1334x750 | @2x | 4.7" |
| iPhone 5 (5S,5C,5) | Oui | 640x1136 | 1136x640 | @2x | 4.0" |
| iPhone 4 (4S,4) | Oui | 640x960 | 960x640 | @2x | 3.5" |
| iPhone (1,2,3) | Non | 320x480 | 480x320 | @1x | 3.5" |
| iPad Air (1,2) / iPad Retina (3,4) | Oui | 1536x2048 | 2048x1536 | @2x | 9.7" |
| iPad Mini (2,3) | Oui | 1536x2048 | 2048x1536 | @2x | 7.9" |
| iPad Mini (1) / iPad (1,2) | Non | 768x1024 | 1024x768 | @1x | 7.9"/9.7" |

Les **pixels** sont les éléments physiques les plus petits que l'on peut contrôler sur un écran. Plus il y a de pixels sur une surface définie, meilleur sera le **PPI** (pixels-per-inch) et plus clair

sera le contenu affiché.

Les **points** sont une mesure de résolution indépendante. En fonction de la densité des pixels sur l'écran, un point peut contenir plusieurs pixels. (exemple : 1 point contient 2 x 2 pixels sur un écran Retina normal).

Quand on développe une application sur différent type d'écrans, *on doit penser en points, mais concevoir en pixels*. Cela signifie que l'on doit exporter toutes les ressources en 3 résolutions différentes, sans se préoccuper de la résolution dans laquelle on conçoit l'application.

## B.4 Icônes d'application

| Appareil | Application | App Store | Spotlight | Réglages |
|---|---|---|---|---|
| iPhone 6+ | 180x180 | 1024x1024 | 120x120 | 87x87 |
| iPhone (6,5S,5,4S,4) | 120x120 | 1024x1024 | 80x80 | 24x24 |
| iPhone (1,2,3) | 57x57 | 1024x1024 | 80x80 | 58x58 |
| iPad Retina (Mini 2,3 / Air / 3,4) | 152x152 | 1024x1024 | 80x80 | 58x58 |
| iPad Retina / iPad (1,2 / Mini 1,2) | 76x76 | 1024x1024 | 40X40 | 29x29 |

## B.5 Typographie

La police par défaut sur *iOS* est **Helvetica Neue**, même si elle est légèrement modifiée depuis *iOS 7*. On peut utiliser toutes les polices *True Type Font* (.ttf), mais il faut faire attention aux licences. Il est également recommandé d'utiliser la même police dans toute l'application à des fins de cohérence et d'uniformité.

## B.6 Couleurs

Depuis *iOS 7*, *Apple* utilise une palette de couleur vive pour ses applications. On peut ainsi utiliser la palette de couleur par défaut (voir figure B.2) ou nos propres couleurs.

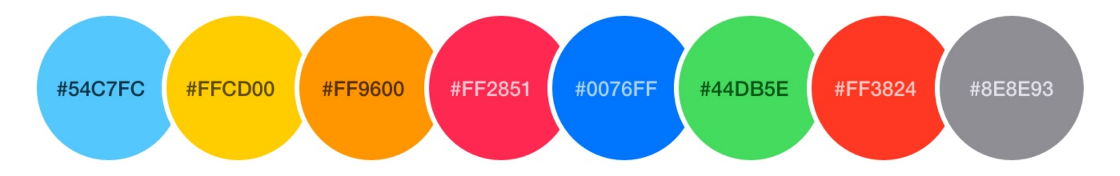



FIGURE B.2 – Palette de couleurs d'Apple

## B.7 Icônes

Dans les applications *iOS*, les icônes sont toujours un excellent moyen d'appuyer les textes grâce à une relation visuelle forte ou en remplacement total de texte. En général, on utilise les icônes dans la barre de navigation ou la barre d'onglets.

Dans les barres, les icônes devraient toujours avoir deux états **inactif**, (gris et non rempli par défaut) ou **actif** (bleu et rempli par défaut). Ils doivent également n'être constitués que d'une seule couleur et posséder un fond transparent (voir figure B.3).

FIGURE B.3 – Icônes actifs/inactifs

## B.8 Éléments d'interface utilisateur

*iOS* offre une grande collection d'éléments prêts à l'emploi qui permet aux développeurs de construire rapidement une interface. Et bien sûr ces éléments peuvent être personnalisés (mais pas tous). Il peut aussi être utile de créer ses propres éléments, car tout est possible, mais il faut toujours bien réfléchir avant de créer un nouvel élément de design.

### B.8.1 Barre de statut, « Status Bar »

Elle contient des informations basiques telles que l'opérateur, l'heure, le statut de la batterie, etc. Elle est visuellement connectée à la barre de navigation. Pour correspondre au design de l'application et accroître la lisibilité, le contenu de la barre de statut peut être sombre (figure B.4) ou clair (figure B.5).

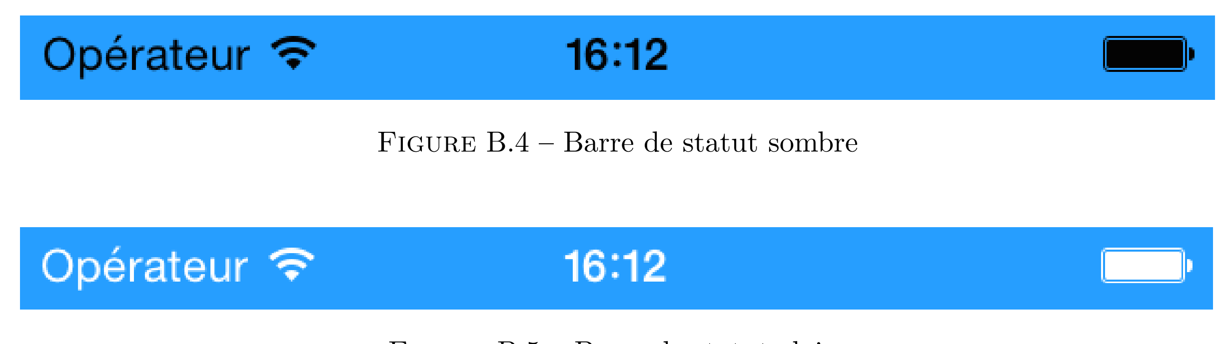



FIGURE B.4 – Barre de statut sombre

FIGURE B.5 – Barre de statut claire

### B.8.2 Barre de navigation, « Navigation Bar »

Elle contient les contrôles pour naviguer à travers l'application et/ou pour gérer le contenu de la vue actuelle. Toujours en haut sous la barre de statut. Par défaut, elle est légèrement transparente (figure B.6).

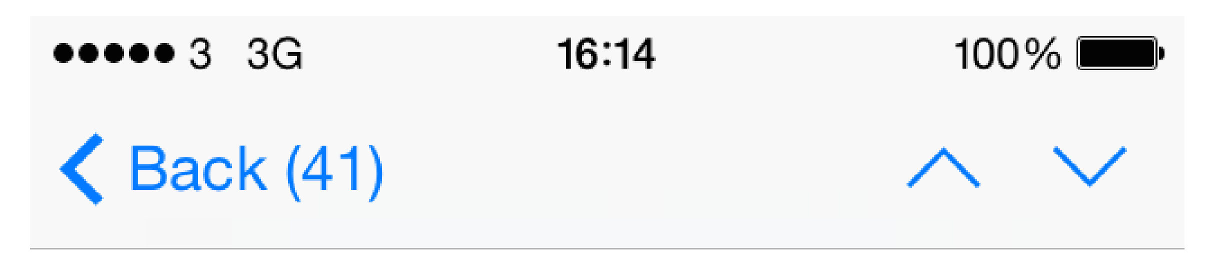



FIGURE B.6 – Barre de navigation

Les éléments devraient ainsi essayer de respecter un maximum les règles suivantes :

- le bouton de retour à gauche
- le titre au centre (non présent sur la figure B.6)
- les boutons d'action à droite

### B.8.3 Barre de recherche, « Search Bar »

Les barres de recherche sont très souvent liées avec le contenu de la vue et permettent de le filtrer. Par défaut, lorsque l'utilisateur active la barre de recherche le focus affiche le clavier.

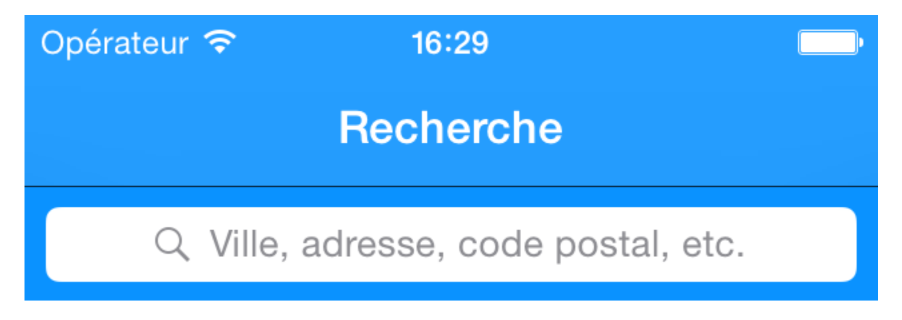



Figure B.7 – Barre de recherche

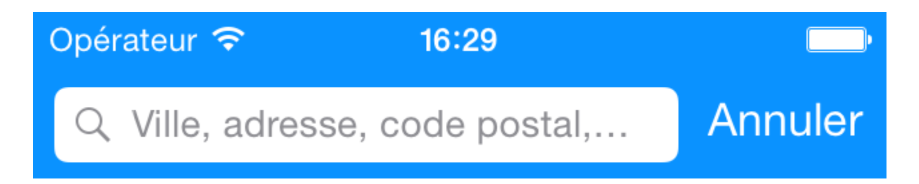



Figure B.8 – Focus dans la barre de recherche

### B.8.4 Barre d'onglets, « Tab Bar »

La barre d'onglets est utilisée pour permettre à l'utilisateur de naviguer rapidement entre les vues de l'application. Elle apparaît souvent en bas de l'écran. Par défaut, elle est légèrement transparente de la même façon que la barre de navigation. Deux styles sont disponibles : sombre et clair (figure B.9).

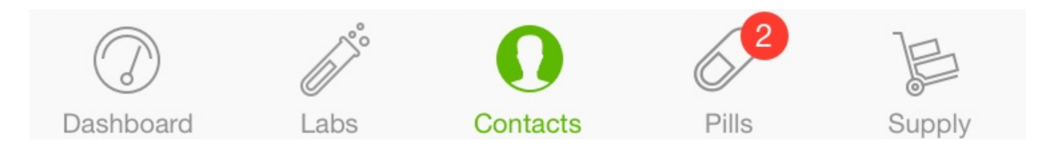



Figure B.9 – Barre d'onglets

Il est possible d'afficher en même temps un maximum de 5 onglets sur iPhone et 7 onglets sur iPad.

### B.8.5 Table, « TableView »

La *TableView* est utilisée pour afficher quelques informations sous forme d'une liste, séparée en section ou non, en groupe ou non, etc.

Le style par défaut propose une image (optionnelle) alignée à gauche et un titre.

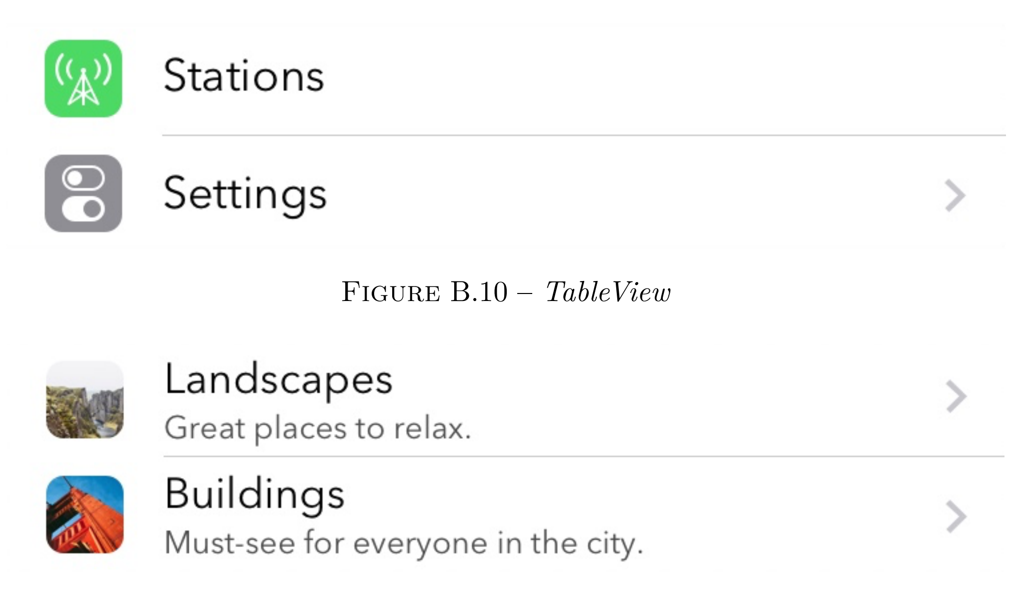



FIGURE B.10 – *TableView*

FIGURE B.11 – *TableView* avec sous-titre

On peut également ajouter un sous-titre, qui peut être utile pour ajouter des explications ou une courte description.

On peut également ajouter une valeur à la place de l'indicateur à droite. Les *TableView* sont entièrement personnalisables.

### B.8.6 Contrôles

*iOS* fournit une large sélection de contrôles pour tout type d'entrée. Les plus importantes sont listées si dessous, mais il en existe beaucoup d'autres.

#### B.8.6.1 Boutons

Sûrement le contrôle le plus utilisé. Depuis *iOS 7*, le bouton défaut ressemble plus à une zone de texte cliquable. Il est personnalisable à volonté, du texte à la couleur en passant par l'image.

#### B.8.6.2 Sélecteurs

Les sélecteurs sont utilisés pour sélectionner une valeur parmi une liste de valeurs disponibles. Les sélecteurs sont très peu modifiables.

FIGURE B.12 – Sélection d'une date

#### B.8.6.3 Sliders

Les sliders sont utilisés pour choisir une valeur parmi une plage de valeurs disponibles. Ils sont utilisés pour choisir une valeur approximative, non exacte.



Figure B.13 – Slider

#### B.8.6.4 Switch

Les switchs permettent de rapidement basculer entre deux états : on et off.

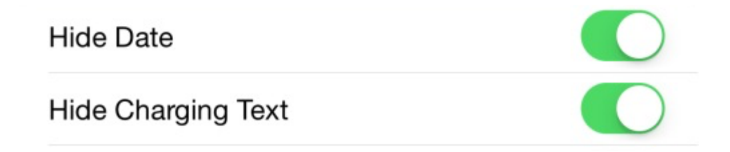



Figure B.14 – Switchs




# Algorithme de récupération de la distance et de la durée de trajet estimée

**Données**: *place* - le marqueur sélectionné
**Résultat**: distance et temps de parcours récupérés

```
/* On défini notre source et notre destination */
sourcePlacemark ← MKPlacemark(coordinate : locationManager.location.coordinate,
addressDictionary : nil)
destinationPlacemark ← MKPlacemark(coordinate : place.position, addressDictionary :
nil)
source ← MKMapItem(placemark : sourcePlacemark)
destination ← MKMapItem(placemark : destinationPlacemark)
/* On défini les paramètres de notre requête */
directionRequest ← MKDirectionsRequest()
directionRequest.setSource(source)
directionRequest.setDestination(destination)
directionRequest.transportType ← MKDirectionsTransportType.Automobile
directionRequest.requestsAlternateRoutes ← vrai
/* On créé notre itinéraire et on effectue la demande */
directions ← MKDirections(request : directionRequest)
directions.calculateDirectionsWithCompletionHandler({
/* On rentre dans notre closure pour traiter les résultats */
   (response : MKDirectionsResponse!, error : NSError?) in
   timeETA ← initialisation
   distanceETA ← initialisation
   si response n'est pas nulle et contient au moins une route alors
      route ← la première route trouvée
      timeETA ← le temps de trajet estimé de la route
      distanceETA ← la distance de trajet estimée de la route
   fin si
   /* On met à jour les données de l'emplacement */
   place.setDistanceAndDurationETA(distanceETA, timeETA)
})
```

# Bibliographie


[1] The World Bank World Heath Organization. World report on disability, 2011. URL `http://whqlibdoc.who.int/hq/2011/WHO_NMH_VIP_11.01_eng.pdf`.

[2] European Commission. EU Parking Card, 2008. URL `http://ec.europa.eu/social/main.jsp?catId=1139`.

[3] IEEE Computer Society. IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications. 1998.

[4] Open Data Commons. Open Database License. 2011.

[5] Mike Stern. Designing Intuitive User Experiences. *WWDC - Session 211*, 2014.

[6] Tim Berners-Lee. The Next Web. *TED*, 2009.

**Résumé**

L'accès au transport est, pour une personne handicapée, un droit qui devrait être acquis depuis longtemps. Mais dans de nombreux cas, les systèmes de transport et l'information ne sont pas accessibles facilement ou peu d'informations sont disponibles. L'application *HandiCarParking* a pour objectif de localiser rapidement les places de stationnement réservées aux personnes handicapées dans le monde entier. Elle se base sur les données d'*OpenStreetMap*. L'application est disponible gratuitement sur l'App Store.

**Mots-clés :** places de parking ; personnes handicapées ; application iOS ; OpenStreetMap

**Abstract**

Access to transportation is for a disabled person, a right that should be acquired for a long time. But in many cases, transport and information systems are not easily accessible or only some information are available. The *HandiCarParking* application aims to quickly locate parking spaces reserved for disabled people in the world. It is based on the data of *OpenStreetMaps*. The application is available for free on the App Store.

**Keywords:** parking spaces ; disabled persons ; iOS application ; OpenStreetMap